Quantum Speedup and Mathematical Solutions of Implementing Bio-molecular Solutions for the Independent Set Problem on IBM Quantum Computers

Weng-Long Chang[1]*, Ju-Chin Chen [2]*, Wen-Yu Chung [3]*, Chun-Yuan Hsiao[4]*, Renata Wong[5]* and Athanasios V. Vasilakos[6]*

***Abstract*— In this paper, we propose a bio-molecular algorithm with $O(n^2 + m)$ biological operations, $O(2^n)$ DNA strands, $O(n)$ tubes and the longest DNA strand, $O(n)$, for solving the independent-set problem for any graph G with m edges and n vertices. Next, we show that a new kind of the straightforward Boolean circuit yielded from the bio-molecular solutions with m *NAND* gates, $(m + n \times (n + 1))$ *AND* gates and $((n \times (n + 1)) / 2)$ *NOT* gates can find the maximal independent-set(s) to the independent-set problem for any graph G with m edges and n vertices. We show that a new kind of the proposed quantum-molecular algorithm can find the maximal independent set(s) with the lower bound $\Omega(2^{n/2})$ queries and the upper bound $O(2^{n/2})$ queries. This work offers an obvious evidence for that to solve the independent-set problem in any graph G with m edges and n vertices, bio-molecular computers are able to generate a new kind of the straightforward Boolean circuit such that by means of implementing it quantum computers can give a quadratic speed-up. This work also offers one obvious evidence that quantum computers can significantly accelerate the speed and enhance the scalability of bio-molecular computers. Next, the element distinctness problem with input of n bits is to determine whether the given 2^n real numbers are distinct or not. The quantum lower bound of solving the element distinctness problem is $\Omega(2^{n\times(2/3)})$ queries in the case of a quantum walk algorithm. We further show that the proposed quantum-molecular algorithm reduces the quantum lower bound to $\Omega((2^{n/2}) / (2^{1/2}))$ queries. Furthermore, to justify the feasibility of the proposed quantum-molecular algorithm, we successfully solve a typical independent set problem for a graph G with two vertices and one edge by carrying out experiments on the backend ibmqx4 with five quantum bits and the backend simulator with 32 quantum bits on IBM's quantum computer.**

***Index Terms*— data structures and algorithms, independent-set problem, molecular algorithms, molecular computing, quantum algorithms, quantum computing**

I. Introduction

FEYNMAN [1] was the first to propose molecular computation without implementing the idea himself. After a few decades, by handling DNA strands, Adleman [2] succeeded in solving an instance of the Hamiltonian path problem in a test tube. In 1982, Feynman [3] raised one of the most important problems in computation theory, namely, whether computing devices based on quantum theory will be able to complete computations faster than the standard Turing machines [4]. Benioff [5] has considered the possibility of quantum computation as well. Deutsch designed a general model of quantum computation – the quantum Turing machine [6].

A graph $G = (V, E)$ is defined in terms of vertices and edges, where V is a set of n vertices and E is a set of m edges. Mathematically, an *independent set* of a graph $G = (V, E)$ is a subset $V^1 \subseteq V$ of vertices such that for every two vertices in V^1, there is no edge connecting the two [7]. The independent-set problem is to find a *maximum-size* independent set in G. This problem is NP-complete [7].

This work was submitted on October 19, 2020.

This work was supported by the National Science Foundation of the Republic of China under MOST 105-2221-E-151-040-.

W.-L. Chang is with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, No. 415, Jiangong Road, Sanmin District, Kaohsiung City, Taiwan 807-78, Republic of China (e-mail: changwl@cc.kuas.edu.tw).

J.-C. Chen is with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, No. 415, Jiangong Road, Sanmin District, Kaohsiung City, Taiwan 807-78, Republic of China (e-mail: jc.chen@nkust.edu.tw).

W.-Y. Chung is with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, No. 415, Jiangong Road, Sanmin District, Kaohsiung City, Taiwan 807-78, Republic of China (e-mail: wychung@nkust.edu.tw).

C.-Y. Hsiao is with the Department of Computer Science and Information Engineering, National Kaohsiung University of Science and Technology, No. 415, Jiangong Road, Sanmin District, Kaohsiung City, Taiwan 807-78, Republic of China (e-mail: cyhsiao@nkust.edu.tw).

R. Wong is with the Department of Computer Science and Technology, Nanjing University, 163 Xianlin Road, 210023 Nanjing, Jiangsu, China (e-mail: renata.wong@protonmail.com).

A. V. Vasilakos is with the School of Electrical and Data Engineering, University of Technology, Sydney, Australia, with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China, and with the Department of Computer Science, Electrical and Space Engineering, Lulea University of Technology, Lulea 97187, Sweden (e-mail:th.vasilakos@gmail.com).

Our major contributions in this paper are as follows.

- We show that the independent-set problem for any graph *G* with *m* edges and *n* vertices can be solved by the proposed molecular algorithm with $O(n^2 + m)$ biological operations, $O(2^n)$ DNA strands, $O(n)$ tubes and the longest DNA strand, $O(n)$.
- We demonstrate that a new kind of the straightforward Boolean circuit obtained from bio-molecular solutions with *m* *NAND* gates, $(m + n \times (n+1))$ *AND* gates and $\left(\frac{n\times(n+1)}{2}\right)$ *NOT* gates can find the maximal independent-set(s) to the independent-set problem for any graph *G* with *m* edges and *n* vertices.
- We show that the proposed quantum-molecular algorithm for implementing a new kind of the straightforward Boolean circuit generated from bio-molecular solutions can give a quadratic speed-up for the same problem. This is the best *known* speed-up because the lower bound of solving the problem is $\Omega\left(2^{n\times\frac{1}{2}}\right)$ queries while the upper bound is $O\left(2^{n\times\frac{1}{2}}\right)$ queries.
- We prove that this work offers an obvious evidence for solving the independent set problem for any graph *G* with *m* edges and *n* vertices. Bio-molecular computers are able to generate a new kind of the straightforward Boolean circuit such that when implemented by quantum computers it can provide a quadratic speed-up, which is the best speed-up known for the given problem.
- We demonstrate that this work offers another obvious evidence that quantum computers can significantly accelerate and enhance the speed and the scalability of bio-molecular circuits.
- We show how the mathematical solutions of the same bio-molecular solutions are encoded in terms of a unit vector in the finite-dimensional Hilbert.
- We also prove that the processing of reduction among NP-complete problems not only cannot speed up the performance of quantum algorithms but, to the contrary, slows it down.
- We demonstrate that reduction among NP-complete problems is useless for a quantum computer and one should therefore independently develop a new quantum algorithm for solving any NP-complete problem with a quadratic speed-up.
- We show that the proposed quantum-molecular algorithm with a quadratic speed-up for solving the independent set problem in a graph *G* with *n* vertices and *m* edges is not the *best* or *optimal* quantum algorithm.
- The element distinctness problem with an input of *n* bits is to determine whether the given 2^n real numbers are distinct or not. A quantum lower bound for solving it is $\Omega\left(2^{n\times\frac{2}{3}}\right)$ queries to a quantum walk algorithm. We show that the proposed quantum-molecular algorithm reduces the quantum lower bound to $\Omega\left(\sqrt{\frac{2^n}{2}}\right)$ queries.
- We experimentally solve an instance of the independent set problem in a graph with two vertices and one edge on the IBM backend *ibmqx4* with five quantum bits and the backend *simulator* with 32 quantum bits.

The rest of the paper is organized as follows: in Section II, the motivation for this work is given. In Section III, we illustrate the development of molecular computers and quantum computers. In Section IV, the molecular algorithm for solving the independent set problem for any graph *G* with *m* edges and *n* vertices is proposed. In Section V, we propose a quantum algorithm for solving the independent set problem on any graph with *m* edges and *n* vertices. In Section VI, we analyze the time complexity and the space complexity of the proposed quantum-molecular algorithm for solving the same problem. In Section VII, we show how the mathematical solutions of the molecular solutions for the same problem are encoded in terms of a unit vector in the finite-dimensional Hilbert space. In Section VIII, we demonstrate that reduction among NP-complete problems is *useless* and independently developing a better quantum algorithm for each NP-complete problem is the right way to approach this issue. In Section IX, we show that the quantum lower bound is $\Omega(2^{n\times\frac{2}{3}})$ queries for a quantum walk algorithm that solves the element distinctness problem with an input of *n* bits. The element distinctness problem is to determine whether the given 2^n real numbers are distinct or not. We give a proof (the reasons) of why it is reduced to $\Omega(\sqrt{\frac{2^n}{2}})$ queries. In Section X, we experimentally solve an instance of the independent set problem in a graph with two vertices and one edge on the IBM backend *ibmqx4* with five quantum bits and the backend *simulator* with 32 quantum bits. In Section XI, we experimentally solve an instance of the independent set problem in a graph with three vertices and two edges on the IBM backend *simulator* with 32 quantum bits. In Section XII, we give a brief conclusion.

II. Motivation

Bennett et al. [8] demonstrated that the lower bound of quantum algorithms for solving any NP-complete problem with input size *n* bits is $\Omega(2^{n\times\frac{1}{2}})$. This result indicates that a new kind of quantum algorithm for solving any NP-complete problem can give a quadratic speed-up that is the *best* speed-up *known* for the problem if its upper bound is $O(2^{n\times\frac{1}{2}})$. An interesting open question is "what are the mathematical solutions of molecular solutions for solving any NP-complete problem"? The independent set problem on any graph with *m* edges and *n* vertices is an NP-complete problem [7, 9], and its molecular solution, its quantum solution and mathematical solution of the same molecular solution haven't yet been proposed. Our motivation for writing this article is to search for the three solutions.

III. The Development of Molecular and Quantum Computers

A potentially significant area of application for DNA algorithms was the breaking of encryption schemes [10-13]. For

solving many well-known computational problems, the proposed DNA algorithms included the 3-SAT problem [14], three-vertex-coloring [15], the binary integer programming problem [16], the subset-production [17] and real DNA experiments of Knapsack problems [18]. Other well-known DNA algorithms include the set partition problem [19], molecular verification of rule-based systems [20], implementation of bio-molecular databases [21] and implementation of arithmetical operations of complex vectors [22]. Woods et al. [23] report the design and experimental validation of a DNA tile set that contains 355 single-stranded tiles and can, through simple tile selection, be reprogrammed to implement a wide variety of 6-bit algorithms. Noteworthy are also recent developments in DNA computing algorithms and models, such as [24-26].

One of the first quantum algorithms was the Deutsch-Jozsa algorithm that showed how to exploit some inherently quantum mechanical features of quantum Turing machines [27]. In 1994, Shor proposed his quantum algorithm for efficiently solving factoring and discrete logarithm problems [28]. And in 1996, Grover's algorithm for searching answer(s) in an unsorted database was proposed in [29]. A detailed description of quantum computation and quantum information was given in [30-32]. Aaronson and Shi in [33] proposed quantum lower bounds of quantum walk algorithms for the collision and the element distinctness problems. Huo and Long in [34] have shown that a single-photon entanglement state can be generated in a simple way in the linear interaction regime, and in the nonlinear interaction regime a scheme for generating squeezed states of microwaves using three-wave mixing in solid-state circuits was proposed. Yang et al. in [35] proposed an implementation of a many-qubit Grover search. Long and Xiao in [36] implemented a NMR quantum information processor with seven quantum bits. Boneh and Lipton in [37] proved that quantum computers are able to break any cryptosystem in quantum polynomial time based on what they refer to as a 'hidden linear form'. Lukac and Perkowski in [38] presented an evolutionary approach to the quantum symbolic logic synthesis and used a genetic algorithm to synthesize quantum circuits. Moylett et al. [39] showed a quantum speedup of the traveling-salesman problem for bounded-degree graphs. Chang et al. [40] solved the problem of finding maximal cliques in graphs with a quantum speedup. Pelofske et al. in [41] demonstrated the large minimum vertex cover problems on a quantum annealer. Arute et al. in [42] determined that their Sycamore processor takes about 200 seconds to sample one instance of a quantum circuit a million times—their benchmarks currently indicate that the equivalent task for a state-of-the-art classical supercomputer would take approximately 10,000 years. An introduction to writing quantum programs to solve real applications on IBM's quantum computers and in quantum processing units appeared in [43-44].

IV. Molecular Algorithms for Solving the Independent Set Problem

In this section, we introduce the definition of the independent set problem for any graph with m edges and n vertices. Next, DNA strands and biological operations proposed in [2] are introduced. They will be applied to design molecular circuits to solve the independent set problem. Then, the time complexity and the space complexity of the proposed molecular algorithm is given. After that, the straightforward Boolean circuit generated from bio-molecular solutions to the independent-set problem on any graph with m edges and n vertices is given. And lastly, we use data dependence analysis to prove that the straightforward Boolean circuit to solve the independent set problem on any graph with m edges and n vertices is the best known for the problem.

A. Definition of the Independent Set Problem

Let G be a graph and $G = (V, E)$, where V is a set of vertices and E is a set of edges in G. We assume that V is $\{v_1, \ldots, v_n\}$ and E is $\{(v_a, v_b) | v_a \text{ and } v_b \text{ are, respectively, elements in } V\}$. We further assume that $|V|$ denotes the number of vertices in V and $|E|$ denotes the number of edges in E. We also suppose that $|V|$ is equal to n and $|E|$ is equal to m. The value of m is at most $((n \times (n-1)) / 2)$. An *independent set* of graph G is a subset $V^1 \subseteq V$ of vertices such that for all $v_a, v_b \in V^1$, the edge (v_a, v_b) is *not* in E [7, 9]. **Definition 4-1** cited in [7, 9] is used to denote the independent set problem of graph G with m edges and n vertices.

Definition 4-1: The independent set problem of graph G with n vertices and m edges is to find a maximum-sized independent set in G.

Consider a graph G^1 that consists of three vertices $\{v_3, v_2, v_1\}$ and two edges $\{(v_1, v_2), (v_1, v_3)\}$. The independent sets in G^1 are $\{\}$ that is an empty set, $\{v_1\}$, $\{v_2\}$, $\{v_3\}$ and $\{v_3, v_2\}$. The maximum-sized independent set for G^1 is $\{v_3, v_2\}$. From [7, 9] we have that finding a maximum-sized independent set is an NP-complete problem. Therefore, we can formulate it as a "computational search" problem.

B. Introduction and Implementation of Biological Molecular Operations

DNA (deoxyribonucleic acid) encodes the genetic information of cellular organisms. It consists of polymer chains that are DNA strands. Synthesizing DNA strands is to order by means of using an automated process. Each strand may be made of a sequence of nucleotides, or bases, attached to a sugar-phosphate "backbone". The four DNA nucleotides are adenine, guanine, cytosine and thymine, commonly abbreviated to A, G, C and T, respectively. Each strand has a 5' end and a 3' end by chemical convention. Because one end of the single strand has a free (i.e., unattached to another nucleotide) 5' phosphate group, and the other has a free 3' deoxyribose hydroxyl group, hence, any single strand has a natural orientation, as introduced in [45].

When two separate single strands bond, this bonding forms the classical double helix of DNA. Bonding occurs by the pairwise attraction of bases: A bonds with T and G bonds with C. The pairs (A, T) and (G, C) are therefore known as complementary base pairs [45]. Also in [45] we have that heating the solution to a temperature determined by the composition of the strand may denature double-stranded DNA into single strands. Heating breaks the hydrogen bonds between complementary strands ((Fig. 4-1) in [45]). Because a $G - C$ pair is joined by three hydrogen bonds, the temperature required to break it is slightly higher than that for an $A - T$ pair, joined by only two hydrogen bonds [45]. This is the most important factor when designing sequences to represent computational

elements.

Annealing is the reverse of melting, whereby cooling a solution of single strands, and allowing complementary strands to bind together ((Fig. 4-1) in [45]). In double-stranded DNA, if one of the single strands contains a discontinuity (i.e., one nucleotide is not bonded to its neighbor) then this may be repaired by DNA ligase [45]. This allows us to create a unified strand from several bound together by their respective complements.

We will use the following bio-molecular operations cited in [2, 45-46] to construct molecular solutions for the independent set problem for any graph with m edges and n vertices. The implementation of the eight biological operations that are specified in **Definition 4-2** through **Definition 4-9** from [45] is described below. Each implementation illustrates only one possible way to complete the computational behavior of one biological operation. In laboratory techniques, future improvements may well yield more efficient and error-resistant implementations of biological operations, but this does not reduce the theoretical power of the model. We simply offer description of the implementation in order to show the feasibility, in principle, of executing biological operations in vitro (that is to say, every biological operation is completely feasible using existing laboratory techniques). All sequences produced to represent bits from a biological standpoint must be checked to ensure that the DNA strands that they encode do not form unwanted secondary structures with one another (i.e., strands remain separate at all times, and only bind together when this is required). We have addressed the problem of strand design for DNA-based computing at length, and we use the methods described in [45] to minimize the possibility of unwanted binding.

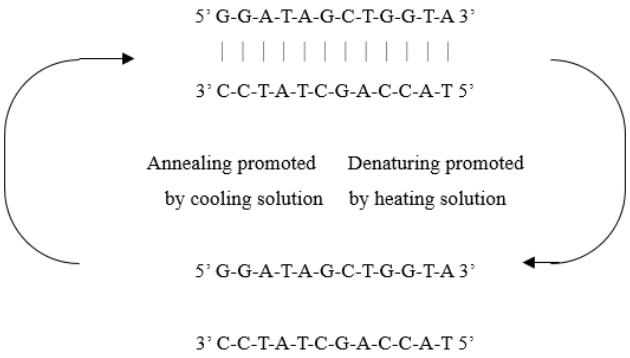

Fig. 4-1. DNA denaturing and annealing.

Definition 4-2: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$ and a bit x_j, the bio-molecular operation "Append-Head" appends x_j onto the head of every element in set X. The formal representation is written as Append-Head(X, x_j) = $\{x_j x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n$ and $x_j \in \{0, 1\}\}$.

Definition 4-3: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$ and a bit x_j, the bio-molecular operation, "Append-Tail", appends x_j onto the end of every element in set X. The formal representation is written as Append-Tail(X, x_j) = $\{x_n x_{n-1} \dots x_2 x_1 x_j | \forall x_d \in \{0, 1\}$ for $1 \le d \le n$ and $x_j \in \{0, 1\}\}$.

Two strands (labeled S and T in Figure 4-2) can be concatenated as follows: create a *linker* strand that includes a sequence that is the complement of S followed by the complement of T. This linker strand is affixed to a surface with a magnetic bead (Figure 4-2(a)). Then, strand S is added to the solution, and anneals with the linker strand in the appropriate position (Fig. 4-2(b)). Then, strand T is added to the solution, and it anneals with the linker strand, at a position immediately adjacent to strand S (Fig. 4-2(c)). Then, we add the ligase enzyme to the solution to seal the "nick" between S and T, forming a single strand that may be freed by heating the solution to break its bonds with the linker strand (Figure 4-2(d)). The implementation of the *concatenate*() operation mentioned above may easily be used to append a specific sequence, s, to the head of each strand in a tube X. In this case, the sequence s corresponds to the strand S shown in Figure 4-2, and strand T in Figure 4-2 corresponds to the beginning sequence of every strand in the tube X. Also, only the starting sequence of every strand anneals to the linker strand. Clearly, then, after a series of *append-head*() operations defined in **Definition 4-2** has been completed on a strand, its sequence will be made up of a number of sequences representing bit-strings. A similar implementation can be applied to complete the *append-tail*() operation defined in **Definition 4-3**.

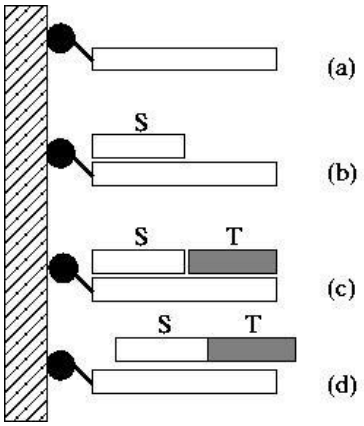

Fig. 4-2. Concatenation process: (a) Linker strand affixed to surface. (b) S anneals to linker strand. (c) T anneals to linker strand, adjacent to S. (d) S and T ligated to form a single strand, which can be freed upon heating the solution.

Definition 4-4: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$, the bio-molecular operation "Discard(X)" resets X to an empty set and can be represented as "$X = \varnothing$".

The *Discard*(X) operation defined in **Definition 4-4** discards the content of a tube X, and replaces the tube X using a new empty tube. Since the number of tubes will generally be one, it is a constant-time operation.

Definition 4-5: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$, the bio-molecular operation "Amplify(X, $\{X_i\}$)" creates a number of identical copies X_i of set X, and then discards X with the help of "Discard(X)".

The *Amplify*(X, $\{X_i\}$) operation defined in **Definition 4-5** is implemented by applying the polymerase chain reaction (PCR) with its initial input being a tube X. This reaction is used to massively amplify (possibly small) amounts of DNA that begin and end with specific primer sequences. Because using these sequences delimits every strand in the tube X, they are all copied

in the reaction. Then, the result of the PCR is equally divided into the specified tubes (therefore, the number of PCR cycles may be adjusted to ensure a constant DNA volume per tube, regardless of the number of tubes).

Definition 4-6: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$ and a bit x_j, the bio-molecular *extract* operation has two kinds of representation. The first representation is $+(X, x_j^1) = \{x_n x_{n-1} \dots x_j^1 \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \ne j \le n\}$ and $-(X, x_j^1) = \{x_n x_{n-1} \dots x_j^0 \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \ne j \le n\}$ if the value of x_j is equal to one. The second representation is $+(X, x_j^0) = \{x_n x_{n-1} \dots x_j^0 \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \ne j \le n\}$and $-(X, x_j^0) = \{x_n x_{n-1} \dots x_j^1 \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \ne j \le n\}$ if the value of x_j is equal to zero.

In order to implement the *extract* operation defined in **Definition 4-6** affinity purification is used to extract any strands from a tube X consisting of a short strand, s, that encodes the value of a bit, x_j. This process uses a probe sequence, which is complementary to the searched target sequence. Probes can attach to a surface and capture strands through annealing any strands consisting of the target sequence. Then, in the rest of the population, separation of captured strands is done by placing them in a separate solution, and then heating the solution in order to break the bonds between the probes and the target sequence. The probe used is therefore the complementary sequence of s. Retained strands are placed in a new tube, $U = +(X, s)$, and the remainder are placed in another new tube, $V = -(X, s)$.

Definition 4-7: Given m sets $X_1 \dots X_m$, the bio-molecular *merge* operation is $\cup(X_1, \dots, X_m) = X_1 \cup \dots \cup X_m$.

The *merge* operation defined in **Definition 4-7** is implemented by pouring the contents of tubes (sets) $\{X_i\}$ into the specified tube. The number of tubes will generally be low, so it is a *constant-time* operation.

Definition 4-8: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$, the bio-molecular operation "Detect(X)" returns *true* if X is not an empty tube. Otherwise, it returns *false*.

The *detect* operation defined in **Definition 4-8** is implemented by running a tube X through a gel electrophoresis process, which is generally used to sort DNA strands by length. Any DNA present in X manifests itself as a visible band in the gel; if DNA strands of the appropriate length are present, the operation returns *true*. If there are no visible bands corresponding to DNA of the correct length, then the operation returns *false*. The length criterion ensures that the present DNA fragments do not cause a false positive result. If the DNA in the band corresponding to the contents of X is required in a subsequent processing step, cutting the band may excise it from the gel. The band is then soaked to remove the strands for further use.

Definition 4-9: Given set $X = \{x_n x_{n-1} \dots x_2 x_1 | \forall x_d \in \{0, 1\}$ for $1 \le d \le n\}$, the bio-molecular operation "Read(X)" describes any element in X. Even if X contains many different elements, this operation can give an explicit description of exactly one of them.

The *read* operation defined in **Definition 4-9** is implemented by using *gel electrophoresis* to sort DNA strands in a tube X by size. Electrophoresis is the movement of charged molecules in an electric field. Because DNA molecules carry a negative charge, they tend to migrate toward the positive pole when placed in an electric field. The rate of migration of a molecule in an *aqueous* solution depends on its shape and electric charge. Because DNA molecules have the same charge per unit length, they all migrate at the same speed in an aqueous solution. However, if a DNA strand completes electrophoresis in a *gel* (usually made of agarose, polyacrylamide or a combination of the two), then its size also affects the migration rate of a molecule. Therefore, the gel is a dense network of pores through which the molecules must travel. Smaller molecules therefore migrate faster through the gel, thus sorting them according to size. DNA strands of the appropriate length in *base pairs* are measured.

C. Molecular Algorithms for Solving the Independent Set Problem

From **Definition 4-1** we have that for any graph G with n vertices and m edges, all possible independent sets are 2^n possible choices containing legal and illegal independent sets in G. Each possible choice corresponds to a subset of vertices in G. Therefore, it is assumed that Y is a set of 2^n possible choices and Y is equal to $\{y_n y_{n-1} \dots y_2 y_1 | \forall y_d \in \{0, 1\}$ for $1 \le d \le n\}$. With this, the length of each element in Y is n bits and each element represents one of the 2^n possible choices. For the sake of presentation, we suppose that y_d^0 indicates that the value of y_d is zero and y_d^1 indicates that the value of y_d is one. If an element $y_n y_{n-1} \dots y_2 y_1$ in Y is a legal independent set and the value of y_d for $1 \le d \le n$ is one, then y_d^1 indicates that the dth vertex is within the legal independent set. If an element $y_n y_{n-1} \dots y_2 y_1$ in Y is a legal independent set and the value of y_d for $1 \le d \le n$ is zero, then y_d^0 indicates that the dth vertex does not appear in the legal independent set.

We propose the following molecular algorithm to solve the independent-set problem for any graph G with n vertices and m edges. The first parameter is an empty tube (a set) Y_0 that is regarded as the input tube (set); the second parameter n represents the number of vertices while the third parameter m represents the number of edges. Each tube in the **Procedure Solve-independent-set-problem**(Y_0, n, m) is an empty tube that is regarded as an auxiliary storage.

Procedure Solve-independent-set-problem(Y_0, n, m)

(0a) Append-Tail(X_1, y_n^1).

(0b) Append-Tail(X_2, y_n^0).

(0c) $Y_0 = \cup(X_1, X_2)$.

(1) **For** $d = n - 1$ **downto** 1

(1a) Amplify(Y_0, X_1, X_2).

(1b) Append-Tail(X_1, y_d^1).

(1c) Append-Tail(X_2, y_d^0).

(1d) $Y_0 = \cup(X_1, X_2)$.

End For

(2) **For** each edge, $e_k = (v_i, v_j)$, in G where $1 \le k \le m$ and bits y_i and y_j respectively represent vertices v_i and v_j.

(2a) $P^1 = +(Y_0, y_i^1)$ and $P^3 = -(Y_0, y_i^1)$.

(2b) $P^2 = +(P^1, y_j^1)$ and $P^4 = -(P^1, y_j^1)$.

(2c) $Y_0 = \cup(P^3, P^4)$.

(2d) Discard(P^2).

End For

(3) **For** $i = 0$ **to** n-1

(4) **For** $j = i$ **down to** 0

(4a) $Y_{j+1}{}^{ON} = +(Y_j, y_{i+1}{}^1)$ and $Y_j = -(Y_j, y_{i+1}{}^1)$.

(4b) $Y_{j+1} = \cup(Y_{j+1}, Y_{j+1}{}^{ON})$.

End For

End For

(5) **For** $c = n$ **down to** 1

(5a) **If** (detect(Y_c)) **then**

(5b) Read(Y_c) and terminate the algorithm.

EndIf

EndFor

EndProcedure

Lemma 4-1: The independent set problem for a graph G with m edges and n vertices can be solved by the molecular algorithm, **Solve-independent-set-problem**(Y_0, n, m).

Proof:

Each execution of Step (0a) and Step (0b), respectively, appends the value "1" for y_n as the first bit of every element in a set X_1 and the value "0" for y_n as the first bit of every element in a set X_2. This indicates that $X_1 = \{y_n{}^1\}$ and $X_2 = \{y_n{}^0\}$. Next, each execution of Step (0c) creates the set union for the two sets X_1 and X_2 so that $Y_0 = X_1 \cup X_2 = \{y_n{}^1, y_n{}^0\}$, and $X_1 = \varnothing$ and $X_2 = \varnothing$.

Next, each execution of Step (1a) creates two identical copies, X_1 and X_2, of set Y_0, and $Y_0 = \varnothing$. Each execution of Step (1b) then appends the value "1" for y_d onto the end of $y_n \ldots y_{d+1}$ for every element in X_1. Similarly, each execution of Step (1c) also appends the value "0" for y_d onto the end of $y_n \ldots y_{d+1}$ for every element in X_2. Next, each execution of Step (1d) creates the set union for the two sets X_1 and X_2 so that $Y_0 = X_1 \cup X_2$, and $X_1 = \varnothing$ and $X_2 = \varnothing$. After repeatedly executing Steps (1a) through (1d), $Y_0 = \{y_n y_{n-1} \ldots y_2 y_1 | \forall y_d \in \{0, 1\}$ for $1 \le d \le n\}$. This is to say that 2^n DNA strands in tube Y_0 encode 2^n possible choices (independent sets).

Next, Step (2) is a loop that evaluates each formula with the form $\left(\overline{y_i \wedge y_j}\right)$ for the kth edge in G where $1 \le k \le m$. On each execution of Step (2a), tube P^1 consists of those DNA strands that have $y_i = 1$, tube P^3 contains those DNA strands that have $y_i = 0$, and tube Y_0 becomes an empty tube. Next, on each execution of Step (2b), tube P^2 contains those DNA strands that have $y_i = 1$ and $y_j = 1$, tube P^4 contains those DNA strands that have $y_i = 1$ and $y_j = 0$, and tube P^1 becomes an empty tube. This indicates that molecular solutions in tube P^2 contain two vertices in the kth edge and are *illegal* independent sets; molecular solutions in tube P^4 only contain one vertex in the kth edge and are *legal* independent sets; and molecular solutions in tube P^3 contain one vertex or no vertices in the kth edge and are *legal* independent sets. Then, on each execution of Step (2c), tube Y_0 contains those DNA strands that encode legal independent sets, tube P^3 is an empty tube, and tube P^4 is also an empty tube. Next, on each execution of Step (2d), *illegal* independent sets encoded by DNA strands in tube P^2 are discarded. After repeatedly executing Steps (2a) through (2d), tube Y_0 consists of those DNA strands that satisfy $\wedge_{k=1}^{m} \left(\overline{y_i \wedge y_j}\right)$ that is the true value for the kth edge in G for $1 \le k \le m$.

Next, Steps (3) and (4) are subsequently the outer loop and the inner loop of the only nested loop, and the range of the *first* loop index variable i is from 0 through $n - 1$, while the range of the *second* loop index variable j is from i down to 0. Each execution of Step (4a) at the iteration (i, j) in the two-level nested loop is applied to compute the influence of y_{i+1} on the number of ones in tubes (sets) Y_{j+1} and Y_j. Upon each execution of Step (4a), the *extract* operation forms two different tubes (sets), $Y_{j+1}{}^{ON}$ and Y_j, from tube (set) Y_j. This is to say that tube (set) $Y_{j+1}{}^{ON}$ has $y_{i+1} = 1$ and tube (set) Y_j has $y_{i+1} = 0$. This indicates that at the iteration (i, j) in the two-level nested loop, the influence of y_{i+1} on the number of ones is to record single ones in tube (set) $Y_{j+1}{}^{ON}$ and also to record zero ones in tube (set) Y_j. Next, upon each execution of Step (4b) at the iteration (i, j) in the two-level nested loop, the *merge* operation is applied to pour the content of tube (set) $Y_{j+1}{}^{ON}$ into tube (set) Y_{j+1}. This indicates that at the iteration (i, j) in the two-level nested loop, the influence of y_{i+1} on the number of ones is to record single ones in tube (set) Y_{j+1}. Next, from the iteration $(i, j - 1)$ through the iteration $(n - 1, 0)$ in the two-level nested loop, similar processing is applied to compute the influence of y_{i+1} through y_n on the number of ones. Hence, after each operation is completed, those DNA strands in tube Y_i for $0 \le i \le n$ have i ones that contain *i vertices*. Next, Step (5) is a loop and it is to read molecular solutions of a maximum-sized independent set. On each execution of Step (5a), if there are DNA strands in tube Y_c, a "true" is returned. Next, on each execution of Step (5b), the answer of a maximum-sized independent set is read and the algorithm terminates. Hence, it is inferred that the independent set problem for a graph G with m edges and n vertices can be solved by the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m). ■

D. Time and Space Complexity of Molecular Algorithms for Solving the Independent Set Problem

The following lemma is used to describe the time complexity, the volume complexity of solution space, the number of the tube used and the longest library strand in solution space for the molecular algorithm, **Solve-independent-set-problem**(Y_0, n, m).

Lemma 4-2. The independent set problem for any graph G with n vertices and m edges can be solved with $O(n^2 + m) = O(n^2)$ biological operations, $O(2^n)$ DNA strands, $O(n)$ tubes and the longest DNA strand, $O(n)$.

Proof:

In the molecular algorithm, **Solve-independent-set-problem**(Y_0, n, m), Steps (0a), (0b) and (0c) take two "Append-Tail" operations and one "Merge" operation. Next, on the execution of Step (1a) through Step (1d), it takes $(n - 1)$ "Amplify" operations, $(2 \times (n - 1))$ "Append-Tail" operations

and $(n - 1)$ "Merge" operations. Next, because the value of m, which is the number of edges, is at most $((n \times (n - 1)) / 2)$, on the execution of Step (2a) through Step (2d), at most $(n \times (n - 1))$ "Extraction" operations, $((n \times (n - 1)) / 2)$ "Merge" operations and $((n \times (n - 1)) / 2)$ "Discard" operations are needed. Next, on the execution of Step (4a) through Step (4b), it takes $((n \times (n + 1)) / 2)$ "Extraction" operations and $((n \times (n + 1)) / 2)$ "Merge" operations. Finally, on the execution of Step (5a) through Step (5b), at most (n) "Detect" operations and one "Read" operation are needed.

From the proof of **Lemma 4-1** we have that the 2^n DNA strands that encode the 2^n possible independent sets are constructed, $(2 \times n + 7)$ tubes are used. Because the length of each possible independent set is n bits and each bit can be encoded by a short DNA strand with *constant* length, the longest DNA strand is $O(n)$. Therefore, from the statements above it is at once inferred that the independent-set problem for any graph G with n vertices and m edges can be solved with $O(n^2 + m) = O(n^2)$ biological operations, $O(2^n)$ DNA strands, $O(n)$ tubes and the longest DNA strand, $O(n)$. ■

E. The Straightforward Boolean Circuit for Determining Independent Sets from Bio-molecular Solutions

After each biological operation from Step (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m) is completed, the 2^n DNA strands in tube Y_0 encode the 2^n possible choices. Next, after each biological operation from Step (2a) through Step (2d) at the same iteration k for $1 \le k \le m$ is completed, bio-molecular solutions in tube P^2 contain two vertices in the kth edge and are *illegal* independent sets while bio-molecular solutions in tube Y_0 contain one vertex or zero vertices in the kth edge and are *legal* independent sets. Therefore, the truth table used to implement the **NAND** operation that appears in Table 4.1 can express the straightforward Boolean circuit generated from Step (2a) through Step (2d) at the same iteration k for $1 \le k \le m$.

Input		Output
y_i	y_j	$\overline{y_i \wedge y_j} = l_k$
0	0	1
0	1	1
1	0	1
1	1	0

Table 4-1: The truth table for the **NAND** operation.

Bits y_i and y_j are its two inputs, and bit l_k for $1 \le k \le m$ is its output. If the value of bit l_k for $1 \le k \le m$ is equal to 1, then the corresponding subsets of vertices only contain one vertex or zero vertices in the kth edge (v_i, v_j) and are legal independent sets. Otherwise, the corresponding subsets of vertices contain two vertices in the kth edge (v_i, v_j) and are illegal independent sets. Therefore, after repeatedly executing Steps (2a) through (2d) from iteration one through iteration m, bio-molecular solutions in tube Y_0 contain one vertex or zero vertices in each edge and do not contain two vertices of any one edge. This is to say that bio-molecular solutions in tube Y_0 encode those subsets of vertices in which for all vertices v_i and v_j, the edge (v_i, v_j) is *not* in E which is the set of edges in graph G. This also implies that bio-molecular solutions in tube Y_0 satisfy the fact that each **NAND** operation of two inputs y_i and y_j has a true value. Therefore, the straightforward Boolean circuit generated from Step (2a) through Step (2d) at all m iterations is to implement the Boolean formula $(\wedge_{k=1}^{m} (\overline{y_i \wedge y_j}))$ and to find which subsets of vertices satisfy the Boolean formula $(\wedge_{k=1}^{m} (\overline{y_i \wedge y_j}))$ that has the true value.

Figure 4-3 shows a flowchart for recognizing independent sets of the independent-set problem for a graph G with n vertices and m edges. In Figure 4-3, in statement S_1, the index variable k of the first loop is set to one (1). Next, in statement S_2, the conditional judgement of the first loop is executed. If the value of k is less than or equal to the value of m, then *next executed* instruction is statement S_3. Otherwise, in statement S_6, an *End* instruction is executed to terminate the task of recognizing independent sets.

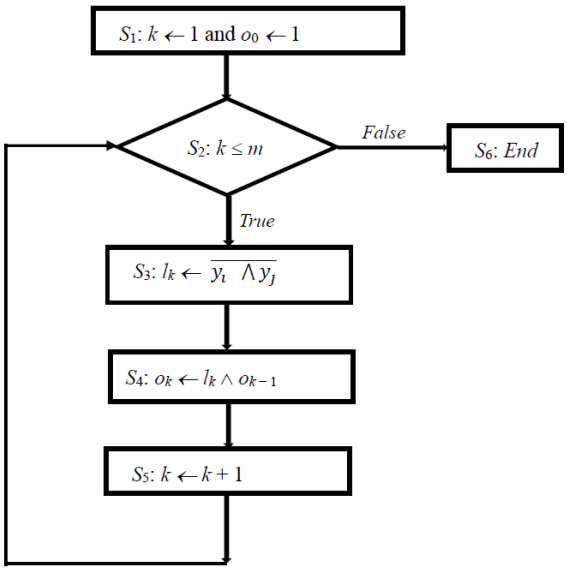

Fig. 4-3: Recognizing independent-sets of the independent-set problem for a graph G with n vertices and m edges.

In statement S_3, a ***NAND*** gate "$l_k \leftarrow \overline{y_i \wedge y_j}$" is implemented. Bits (Boolean variables) y_i and y_j respectively encode vertex v_i and vertex v_j that are connected by the kth edge in a graph G with n vertices and m edges. Bit (Boolean variable) l_k with $1 \le k \le m$ stores the result of implementing $(\overline{y_i \wedge y_j})$ (the kth ***NAND*** gate). Next, in statement S_4, a logical and operation "$o_k \leftarrow l_k \wedge o_{k-1}$" is executed that is the *kth* clause in $(\wedge_{k=1}^{m}(\overline{y_i \wedge y_j}))$. Bit (Boolean variable) l_k stores the result of implementing the kth ***NAND*** gate and is the first operand of the logical and operation. Bit (Boolean variable) o_{k-1} with $1 \le k \le m$ is the second operand of the logical and operation and stores the result of the previous logical and operation. Bit (Boolean variable) o_k with $1 \le k \le m$ stores the result of implementing $l_k \wedge o_{k-1}$ (the kth clause that is the kth ***AND*** gate). Next, in statement S_5, the value of the index variable k to the first loop is incremented. Repeat to execute statements S_2 through S_5 until in statement S_2 the conditional judgement results a *false* value. From Figure 4-3 it follows that the total number of ***NAND*** gates is m. The total number of logical and operation is m ***AND*** gates. Therefore, the

cost of recognizing independent set(s) corresponds to *m* ***NAND*** gates and *m* ***AND*** gates.

A data dependence arises from two statements that access or modify the same resource. *Data dependence analysis* is to determine whether it is safe to *reorder* or *parallelize* statements. If the first of two statements first modifies the same resource and then the second of two statements reads the same resource, then there is a ***true dependence*** between the first statement and the second statement. If the first of two statements first reads the same resource and then the second of two statements modifies the same resource, then there is an ***anti-dependence*** between the first statement and the second statement. If the first of two statements first modifies the same resource and then the second of two statements modifies the same resource, then there is an ***output dependence*** between the first statement and the second statement. We use data dependence analysis to show that the straightforward Boolean circuit in Figure 4-3 for recognizing independent sets of the independent set problem for a graph *G* with *n* vertices and *m* edges is the *best* Boolean circuit *known* for the problem.

Lemma 4-3. For the independent-set problem for any graph G with *n* vertices and *m* edges, in Figure 4-3, the Boolean circuit with *m* ***NAND*** gates and *m* ***AND*** gates generated from Step (2a) through Step (2d) at all *m* iterations in the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*), is the best Boolean circuit known for recognizing independent-set(s) among 2^n possible choices.

Proof:

In Figure 4-3, in statement S_3, a ***NAND*** gate "$l_k \leftarrow \overline{y_t \wedge y_j}$" is implemented and the result of implementing $(\overline{y_t \wedge y_j})$ is written into Boolean variable l_k. Next, in statement S_4, a logical and operation "$o_k \leftarrow l_k \wedge o_{k-1}$" is executed and it needs to read the value of Boolean variable l_k (the first operand). Therefore, there is a ***true dependence*** between statement S_3 and statement S_4. The ***true dependence*** between statement S_3 and statement S_4 cannot be broken. Therefore, each statement in Figure 4-3 must be executed in sequential mode. Hence, it is at once inferred from the statements above that in Figure 4-3 the straightforward Boolean circuit with *m* ***NAND*** gates and *m* ***AND*** gates generated from Step (2a) through Step (2d) at all *m* iterations in the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*), is the best Boolean circuit known for recognizing independent-set(s) among 2^n possible choices. ■

F. The Straightforward Boolean Circuit for Computing the Number of Vertex in Independent Sets from Biomolecular Solutions

After each biological operation from Steps (2a) through (2d) in the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*) is completed, the DNA strands in tube Y_0 encode each solution (independent-set) that has the value of o_m equal to one (1). For computing the number of vertices, we need auxiliary Boolean variables $w_{i+1,j}$ and $w_{i+1,j+1}$ with $0 \le i \le n-1$ and $0 \le j \le i$. Auxiliary Boolean variables $w_{i+1,j}$ and $w_{i+1,j+1}$ with $0 \le i \le n-1$ and $0 \le j \le i$ are set to the initial value 0 (zero). Boolean variable $w_{i+1,j+1}$ with $0 \le i \le n-1$ and $0 \le j \le i$ stores the number of vertex in a solution after figuring out the influence of Boolean variable y_{i+1} that encodes the $(i+1)$th vertex on the number of ones (vertices). If the value of Boolean variable $w_{i+1,j+1}$ for $0 \le i \le n-1$ and $0 \le j \le i$ is equal to 1 (one), then this indicates that there are $(j+1)$ ones (vertices) in the solution. Boolean variable $w_{i+1,j}$ for $0 \le i \le n-1$ and $0 \le j \le i$ stores the number of vertex in a solution after figuring out the influence of Boolean variable y_{i+1} that encodes the $(i+1)$th vertex on the number of ones (vertices). If the value of Boolean variable $w_{i+1,j}$ for $0 \le i \le n-1$ and $0 \le j \le i$ is equal to 1 (one), then this indicates that there are j ones (vertices) in the solution.

In a solution (an independent-set) that has the value of bit o_m equal one, bit y_1 encodes the first vertex v_1. If the value of bit y_1 is equal to one (1), then the first vertex v_1 appears in the solution and it increments the number of vertices (the number of ones) for the solution. Otherwise, the first vertex v_1 does not appear in the solution and it preserves the number of vertices (the number of ones) for the solution. In the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*), on the execution of Step (4a) in the iteration ($i = 0$, $j = 0$), the *extract* operation is used to form two different tubes, Y_1^{ON} and Y_0 out of tube (set) Y_0. Therefore, the DNA strands in tube Y_1^{ON} encode solutions that have $y_1 = 1$ and contain vertex v_1 and the DNA strands in tube Y_0 have $y_1 = 0$ and do not contain vertex v_1. This is to say that the influence of y_1 (the influence of vertex v_1) on the number of ones (the number of vertices) is recorded as single ones in tube Y_1^{ON} and to record zero ones in tube Y_0. Next, on the execution of Step (4b) in the same iteration ($i = 0$, $j = 0$), the *merge* operation is applied by pouring the contents of tube Y_1^{ON} into tube Y_1. This indicates that in the iteration ($i = 0$, $j = 0$), the influence of y_1 on the number of ones is recorded as single ones in tube (set) Y_1. Therefore, for the influence of the first vertex v_1, incrementing the number of vertices in each solution is to satisfy the formula $(o_m \wedge y_1)$ and preserving the number of vertices is to satisfy the formula $(o_m \wedge \overline{y_1})$.

Similarly, the influence of the $(i+1)$th vertex with $1 \le i \le n-1$ is to decide whether in each solution the number of vertices (the number of ones) is incremented or is preserved. In order to increment the number of vertices (the number of ones) in each solution two conditions must be satisfied. The *first* condition is that the $(i+1)$th vertex is within the solution and the *second* condition is that each solution currently has j vertices. In order to preserve the number of vertices (the number of ones) in each solution two conditions must be satisfied. The *first* condition is that the $(i+1)$th vertex is not within the solution and the *second* condition is that each solution currently also has j vertices. Next, on each execution of Step (4a) in the iteration (i, j), the *extract* operation is used to form two different tubes (sets), Y_{j+1}^{ON} and Y_j out of tube (set) Y_j. Hence, the DNA strands in tube Y_{j+1}^{ON} encode each solution that has $y_{i+1} = 1$ and contains vertex v_{i+1}. The DNA strands in tube Y_j on the other hand encode each solution that has $y_{i+1} = 0$ and does not contain vertex v_{i+1}. This indicates that in the iteration (i, j), the influence of y_{i+1} on the number of ones (the number of vertices) is recorded as $(j+1)$ ones in tube Y_{j+1}^{ON} and also as j ones in tube Y_j. Next, on each execution of Step (4b) in the iteration (i, j), the *merge* operation is applied by pouring the contents of tube (set) Y_{j+1}^{ON} into tube

(set) Y_{j+1}. This indicates that in the iteration (i, j), the influence of y_{i+1} on the number of ones (the number of vertices) is recorded as having $(j + 1)$ ones in tube Y_{j+1}. Therefore, for the influence of the $(i + 1)$th vertex for $1 \le i \le n - 1$ in each solution, the two conditions for incrementing the number of vertices (the number of ones) in each solution are to satisfy the Boolean formula $(y_{i+1} \wedge w_{i,j})$. The two conditions for preserving the number of vertices in each solution are to satisfy the Boolean formula $((\overline{y_{i+1}}) \wedge w_{i,j})$.

Fig. 4-4 shows the logical flowchart for counting the number of vertices in each solution. In Fig. 4-4, in statement S_1, a logical and operation "$w_{1,1} \leftarrow o_m \wedge y_1$" is implemented that corresponds to one ***AND*** gate. Boolean variable $w_{1,1}$ stores the result of implementing one ***AND*** gate $(o_m \wedge y_1)$. If the value of $w_{1,1}$ is equal to 1 (one), the number of vertices is incremented so that the number of vertices in each solution with the first vertex v_1 is one. Next, in statement S_2, a logical and operation "$w_{1,0} \leftarrow o_m \wedge \overline{y_1}$" is implemented that corresponds to one ***AND*** gate. Boolean variable $w_{1,0}$ stores the result of implementing one ***AND*** gate $(o_m \wedge \overline{y_1})$. If the value of $w_{1,0}$ is equal to 1 (one), then the number of vertices is preserved so that the number of vertices in each solution *without* the first vertex v_1 is zero.

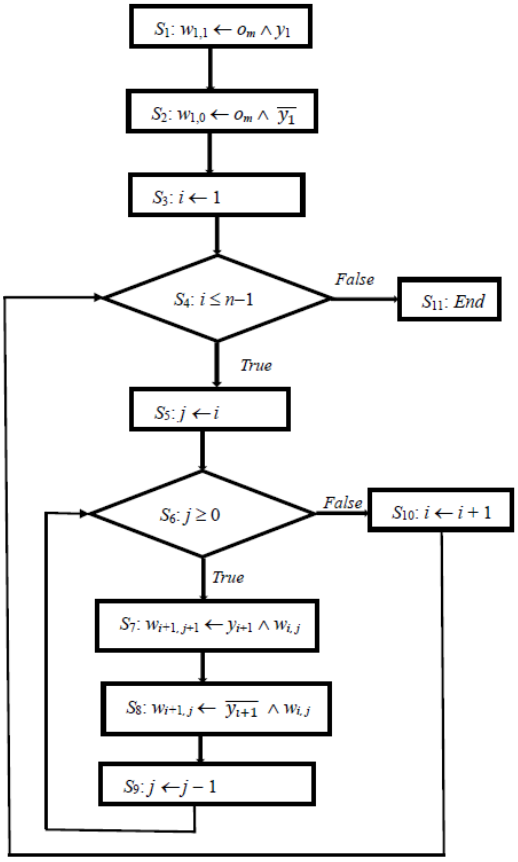

Fig. 4-4: Flowchart for computing the number of vertices in each solution (independent-set).

Next, in statement S_3, the index variable i of the first loop is set to one. Then, in statement S_4, the conditional judgement of the first loop is executed. If the value of i is less than or equal to the value of $(n - 1)$, then *next executed* instruction is statement S_5. Otherwise, in statement S_{11}, an *End* instruction is executed to terminate the task of counting the number of vertices in each solution. In statement S_5, the index variable j of the second loop is set to the value of the index variable i in the first loop. Next, in statement S_6, the conditional judgement of the *second* loop is executed. If the value of j is greater than or equal to zero, then the next executed instruction is statement S_7. Otherwise, the next executed instruction is statement S_{10}.

In statement S_7, a logical and operation "$w_{i+1,j+1} \leftarrow y_{i+1} \wedge w_{i,j}$" is implemented that corresponds to one ***AND*** gate. Boolean variable y_{i+1} encodes the $(i + 1)$th vertex and is the first operand of the logical and operation. Boolean variable $w_{i,j}$ is the second operand of the logical and operation. Boolean variable $w_{i,j}$ stores the number of vertex in a solution after determining the influence of Boolean variable y_i that encodes the ith vertex on the number of ones (vertices). If the value of $w_{i,j}$ is equal to 1 (one), then this indicates that there are j ones (vertices) in the solution. Boolean variable $w_{i+1,j+1}$ stores the result of implementing the logical and operation "$w_{i+1,j+1} \leftarrow y_{i+1} \wedge w_{i,j}$". This is to say that $w_{i+1,j+1}$ stores the number of vertex in a solution after determining the influence of Boolean variable y_{i+1} that encodes the $(i + 1)$th vertex on the number of ones (vertices). If the value of $w_{i+1,j+1}$ is equal to 1 (one), then this implies that there are $(j + 1)$ ones (vertices) in the solution.

Next, in statement S_8, a logical and operation "$w_{i+1,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,j}$" is implemented that corresponds to one ***AND*** gate. Boolean variable y_{i+1} encodes the $(i + 1)$th vertex and its negation $\overline{y_{i+1}}$ is the first operand of the logical and operation. Boolean variable $w_{i,j}$ is the second operand of the logical and operation. It stores the number of vertex in a solution after determining the influence of Boolean variable y_i that encodes the ith vertex on the number of ones (vertices). If the value of $w_{i,j}$ is equal to 1 (one), then this indicates that there are j ones (vertices) in the solution. Boolean variable $w_{i+1,j}$ stores the result of implementing the logical and operation "$w_{i+1,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,j}$". This indicates that $w_{i+1,j}$ stores the number of vertex in a soution after determining the influence of Boolean variable y_{i+1} that encodes the $(i + 1)$th vertex on the number of ones (vertices). The value of $w_{i+1,j}$ being equal to 1 (one) indicates that there are j ones (vertices) in the solution.

Next, in statement S_9, the value of the index variable j in the second loop is decremented. Repeat to execute statement S_6 through statement S_9 until in statement S_6 the conditional judgement attains a *false* value. Next, in statement S_{10}, the value of the index variable i in the first loop is incremented. Repeat to execute statements S_4 through S_{10} until in S_4 the conditional judgement attains a *false* value. When this happens, the next executed statement is S_{11}. In S_{11}, an *End* instruction is executed to terminate the task of counting the number of vertices in each solution. The cost of each operation in Fig. 4-4 is $(n \times (n + 1))$ ***AND*** gates and $(\frac{n\times(n+1)}{2})$ ***NOT*** gates. Therefore, the cost of counting the number of vertices for each solution is to implement $(n \times (n + 1))$ ***AND*** gates and $(\frac{n\times(n+1)}{2})$ ***NOT*** gates. We use data dependence analysis to show that in Fig. 4-4 the straightforward Boolean circuit for counting the number of

vertices in each solution is the *best* Boolean circuit *known* for the problem.

Lemma 4-4. In Fig. 4-4, the Boolean circuit with $(n \times (n+1))$ ***AND*** gates and $(\frac{n\times(n+1)}{2})$ ***NOT*** gates generated from Steps (4a) through (4b) in each iteration in the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*), is the *best* Boolean circuit *known* for counting the number of vertices in each solution.

Proof:

As shown in Figure 4-4, in statement S_7, an ***AND*** gate "$w_{i+1, j+1} \leftarrow y_{i+1} \wedge w_{i, j}$" is implemented and the result of implementing $(y_{i+1} \wedge w_{i, j})$ is written into Boolean variable $w_{i+1, j+1}$ for $1 \le i \le (n-1)$ and $i \ge j \ge 0$. In iteration $(i = 1, j = 1)$ in statement S_7 the value of Boolean variable $w_{2, 2}$ is modified and later, in iteration $(i = 2, j = 2)$ in statement S_7, the value of $w_{2, 2}$ is read. There are similar cases of modifying and reading the same resource in Statement S_7 in later iterations. Hence, there is a ***true dependence*** in statement S_7. Next, in statement S_8, a logical and operation "$w_{i+1, j} \leftarrow \overline{y_{i+1}} \wedge w_{i, j}$" is executed and the result of implementing $(\overline{y_{i+1}} \wedge w_{i, j})$ is written into Boolean variable $w_{i+1, j}$ with $1 \le i \le (n-1)$ and $i \ge j \ge 0$. In iteration $(i = 1, j = 1)$ in statement S_8 the value of Boolean variable $w_{2, 1}$ is modified and later, in iteration $(i = 2, j = 1)$ in statement S_8, the value of $w_{2, 1}$ is read. There are similar cases of modifying and reading the same resource in Statement S_8 in later iterations. Therefore, there is a ***true dependence*** in statement S_8. The ***true dependence*** in both statements S_7 and S_8 cannot be broken.

Next, in iteration $(i = 1, j = 1)$ in Statement S_7 the value of Boolean variable $w_{2, 2}$ is modified and later, in iteration $(i = 2, j = 2)$ in Statement S_8, the value of $w_{2, 2}$ is read. There are similar cases of modifying and reading the same resource between Statements S_7 and S_8 in later iterations. Hence, there is a ***true dependence*** between S_7 and S_8. Next, in iteration $(i = 1, j = 1)$ in Statement S_8 the value of Boolean variable $w_{2, 1}$ is modified and later, in iteration $(i = 2, j = 1)$ in Statement S_7, the value of $w_{2, 1}$ is read. There are similar cases of modifying and reading the same resource between Statements S_8 and S_7 in later iterations. Hence, there is a ***true dependence*** between S_8 and S_7. Next, in iteration $(i = 1, j = 1)$ in Statement S_8 the value of Boolean variable $w_{2, 1}$ is written and later in iteration $(i = 1, j = 0)$ in Statement S_7 the value of $w_{2, 1}$ is read. There are similar cases of modifying the same resource between Statements S_8 and S_7 in later iterations. Hence, there is an ***output dependence*** between Statements S_8 and S_7. This indicates that there are simultaneously two ***true dependences*** and one ***output dependence*** between statements S_7 and S_8. The two ***true dependences*** and the ***output dependence*** between statement S_7 and statement S_8 cannot be broken. Therefore, only sequential mode can be used in each statement in Figure 4-4. From the statements above it is at once derived that in Fig. 4-4, the Boolean circuit with $(n \times (n+1))$ ***AND*** gates and $(\frac{n\times(n+1)}{2})$ ***NOT*** gates generated in Step (4a) through Step (4b) in each iteration in the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n*, *m*), is the best Boolean circuit known for counting the number of vertices in each solution. ■

V. Quantum Algorithms for Implementing the Straightforward Boolean Circuits from Molecular Solutions for Solving the Independent Set Problem

In this section, we introduce quantum bits and quantum gates. Then, we use them to design a new kind of quantum algorithm to implement the straightforward Boolean circuits generated from molecular solutions for solving the independent set problem on any graph with *m* edges and *n* vertices.

A. *Introduction to Quantum Bits and Quantum Gates*

In the two-dimensional Hilbert space [30-32, 43-44], a quantum bit has two computational basis vectors |0> and |1>, and corresponds to the classical bit values 0 and 1. We refer to a collection of *n* quantum bits as a quantum register of size *n*. A quantum register may consist of any of the 2^n-dimensional computational basis vectors, *n* quantum bits of size, or an arbitrary superposition of these vectors [30-32, 43-44]. If the content of the quantum bits of a quantum register is known, then the state of the quantum register can be computed by a tensor product in the following way: $|\partial\rangle = (|q_n\rangle \otimes |q_{n-1}\rangle \otimes \dots \otimes |q_2\rangle \otimes |q_1\rangle)$. If the state of a quantum register of size *n* is an arbitrary superposition of the 2^n-dimensional computational basis vectors, then it can be represented as $|\gamma\rangle = (\sum_{a=0}^{2^n-1} b_a |a\rangle)$, where each weighted factor $b_a \in \mathbf{C}$ is a so-called probability amplitude; hence they must satisfy $(\sum_{a=0}^{2^n-1} |b_a|^2) = 1$.

Unitary operators are often referred to as quantum gates [30-32, 43-44]. Using quantum gates one can model the time evolution of the states of quantum registers. Hence, a quantum gate is an elementary quantum-computing device that completes a fixed unitary operation on selected quantum bits during a fixed period. As given in [30-32, 43-44], the **Hadamard** gate *H* is a quantum gate of one quantum bit (a 2×2 matrix). Its four entries are, respectively, $H_{1, 1} = 1 / (2^{1/2})$, $H_{1, 2} = 1 / (2^{1/2})$, $H_{2, 1} = 1 / (2^{1/2})$, and $H_{2, 2} = -1 / (2^{1/2})$. The **NOT** gate with one quantum bit sets only the (target) bit to its negation. The **CNOT** (*controlled*-**NOT**) gate with two quantum bits flips the second quantum bit (the target quantum bit) if and only if the first quantum bit (the control quantum bit) is equal to one. The *controlled*-*controlled*-**NOT** (**CCNOT)** gate with three quantum bits flips the third quantum bit (the target quantum bit) if and only if the first and second quantum bits (the two control quantum bits) are both one. A quantum gate, $H^{\otimes n}$, that stands for the joined Hadamard gates of *n* quantum bits is applied to an initial state vector $|000\dots0\rangle$ with *n* quantum bits, and its outcome is $|\lambda\rangle = (\frac{1}{\sqrt{2^n}} \sum_{a=0}^{2^n-1} |a\rangle)$.

B. *Computational State Space of Molecular Solutions for the Independent Set Problem*

Based on the molecular algorithm, **Solve-independent-set-problem**(Y_0, *n, m*), 2^n possible choices (independent sets) are generated in Steps (0a) through (1d), and are stored in set (tube) Y_0 which is equal to $\{y_n y_{n-1} \dots y_2 y_1 | \forall y_d \in \{0, 1\}$ for $1 \le d \le n\}$. We use the following lemma to describe computational state space of molecular solutions for solving the independent-set problem for a graph *G* with *n* vertices and *m* edges.

Lemma 5-1: For solving the independent-set problem on a graph with *m* edges and *n* vertices, the set of the corresponding computational state vectors of 2^n possible choices (independent

sets) generated in Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) forms an orthonormal basis of a 2^n dimensional Hilbert space (a complex vector space, C^{2^n}).

Proof:

We use a unique *computational basis vector* with 2^n-tuples of binary numbers to represent each element in set (tube) Y_0. The first corresponding computational basis vector for the *first* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^0$ is $([1 \quad 0 \quad \cdots \quad 0]^T_{1\times 2^n})$. The second computational basis vector for the *second* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^1$ is $([0 \quad 1 \quad \cdots \quad 0]^T_{1\times 2^n})$. And so on, with the *last* corresponding computational basis vector for the *last* element $y_n{}^1\, y_{n-1}{}^1 \ldots y_2{}^1\, y_1{}^1$ being $([0 \quad 0 \quad \cdots \quad 1]^T_{1\times 2^n})$. Therefore, the set of the corresponding computational basis vectors for each element (each possible independent set) in set (tube) Y_0 is $D = \{[1 \quad 0 \quad \cdots \quad 0]^T_{1\times 2^n}, [0 \quad 1 \quad \cdots \quad 0]^T_{1\times 2^n}, \ldots, [0 \quad 0 \quad \cdots \quad 1]^T_{1\times 2^n}\}$. Each computational basis vector in D is a coordinated vector [27], and the vectors together span $D = C^{2^n}$. Therefore, it is at once inferred that the set of the corresponding computational state vectors of 2^n possible choices (independent sets) generated in Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) forms an *orthonormal* basis of a 2^n dimensional Hilbert space (a complex vector space, C^{2^n}). ■

C. *Quantum Circuits and Mathematical Solutions for Computational State Space of Molecular Solutions for the Independent Set Problem*

In light of **Lemma 5-1**, for solving the independent-set problem of a graph with *m* edges and *n* vertices, 2^n possible molecular solutions generated in Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) form an orthonormal basis of a Hilbert space (a complex vector space, C^{2^n}). This is to say that each possible molecular solution corresponds to an element in an orthonormal basis of a Hilbert space (C^{2^n}). For simultaneously encoding 2^n possible molecular solutions, we assume that a quantum register of n bits, $(\otimes_{p=n}^{1}|y_p\rangle)$, is applied to initialize a system that has $Q = 2^n$ states which are labeled as $P_0, P_1, P_2, \ldots, P_{Q-1}$, where each state P_k for $0 \le k \le 2^n - 1$ corresponds to the kth possible molecular solution. We also assume that a quantum register with one quantum bit, $(|1\rangle)$, is used to label the amplitude of the answer(s) among the 2^n states. For completing the purpose, we use one Hadamard gate on the state $|1\rangle$ and the new quantum state vector is $(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle))$.

The initial states in $(\otimes_{p=n}^{1}|y_p\rangle)$ are set to $(\otimes_{p=n}^{1}|y_p^0\rangle)$ and we assume that $|\lambda_0\rangle = (\otimes_{p=n}^{1}|y_p^0\rangle)$. We also suppose that the initial quantum state vector is $(|\lambda_0\rangle)$. Using *n* Hadamard gates to operate on the initial quantum state vector $(|\lambda_0\rangle)$, the system that has $Q = 2^n$ states which are labeled as $P_0, P_1, P_2, \ldots, P_{Q-1}$ is

$$|\lambda_{5-1}\rangle = (H^{\otimes n})\ |\lambda_0\rangle = \frac{1}{\sqrt{2^n}}\left(\otimes_{p=n}^{1}\left(|y_p^0\rangle + |y_p^1\rangle\right)\right) = \frac{1}{\sqrt{2^n}}\left(\sum_{y=0}^{2^n-1}|y\rangle\right). \quad (5\text{-}1)$$

In the new state vector $(|\lambda_{5-1}\rangle)$, state $|y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^0\rangle$ with the amplitude $(\frac{1}{\sqrt{2^n}})$ encodes the *first* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^0$ of molecular solution space that does not contain any vertices. State $|y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^1\rangle$ with the amplitude $(\frac{1}{\sqrt{2^n}})$ encodes the *second* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^1$ of molecular solution space containing the first vertex v_1. And so on, with state $|y_n{}^1\, y_{n-1}{}^1 \ldots y_2{}^1\, y_1{}^1\rangle$ with the amplitude $(\frac{1}{\sqrt{2^n}})$ encoding the *last* element $y_n{}^1\, y_{n-1}{}^1 \ldots y_2{}^1\, y_1{}^1$ of molecular solution space containing *n* vertices $\{v_n\, v_{n-1} \ldots v_2\, v_1\}$.

D. *Quantum Circuits and Mathematical Solutions for Implementing Molecular Solutions for legal Independent Sets among 2^n Possible Choices*

To solve an instance of the independent set problem for a graph *G* with *n* vertices and *m* edges, in Figure 4-3, the straightforward Boolean circuit generated in Steps (2a) through (2d) in all *m* iterations in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) is used to recognize and to label independent-sets among the 2^n possible choices. The straightforward Boolean circuit for labelling legal independent sets among the 2^n possible choices in Figure 4-3 is

$$(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J})), \quad (5\text{-}2)$$

where bits y_i and y_j respectively represent vertices v_i and v_j in the *k*th edge, $e_k = (v_i, v_j)$, in *G* for $1 \le k \le m$. The Boolean formula $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ consists of *m* **NAND** operations and *m* **AND** operations. The operations **NAND** and **AND** are, respectively, implemented by quantum circuits in Figures 5-1(a) and 5-1(b). Therefore, we assume that the second quantum register with *m* quantum bits, $|l_m\, l_{m-1} \ldots l_1\rangle$, for $1 \le k \le m$, stores the result of evaluating the *k*th **NAND** gate with the form $(\overline{y_\iota \wedge y_J})$ that corresponds to one **NAND** operation. The initial state for each quantum bit in $|l_m\, l_{m-1} \ldots l_1\rangle$ is prepared in state $|1\rangle$. The *m*th quantum bit l_m in the second quantum register stores the result of the evaluating computation for the *last* **NAND** operation.

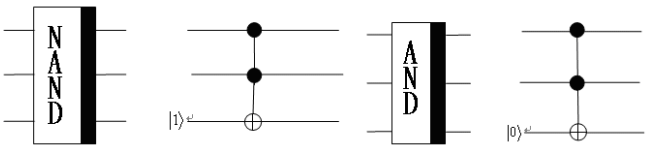

Fig. 5-1: (a) **NAND** operation of two Boolean variables, and (b) **AND** operation of two Boolean variables.

Next, in order to evaluate the **AND** operation of the previous clause (the $(k-1)$th clause) and the current clause (the *k*th clause), a third quantum register $|o_m\, o_{m-1} \ldots o_1\, o_0\rangle$ is needed. The first quantum bit $|o_0\rangle$ in the third quantum register is initially prepared in state $|1\rangle$. Other *m* bits in the third quantum register are initially in state $|0\rangle$. The $(m+1)$th quantum bit $|o_m\rangle$ in the third quantum register stores the result of the evaluation of the **AND** operation of the previous clause (the $(m-1)$th clause) and the *last* clause (the *m*th clause). This indicates that the $(m+1)$th quantum bit $|o_m\rangle$ in the third register stores the result of the evaluating computation for all of the clauses. We use **Lemma 5-2** to show how the quantum circuit in Figure 5-2 implements the straightforward Boolean circuit in equation (5-2) for recognizing legal independent-sets among 2^n possible choices.

Lemma 5-2: To solve the independent set problem for any

graph $G = (V, E)$ with n vertices and m edges, the quantum circuit, **LIS**, in Figure 5-2 with $(2 \times m)$ ***CCNOT*** gates can implement the straightforward Boolean circuit $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ in equation (5-2) and is the *best* quantum circuit *known* for labelling legal independent sets among 2^n possible choices.

Proof:

To solve an instance of the independent set problem for a graph G with n vertices and m edges, in Figure 4-3, the straightforward Boolean circuit generated in Steps (2a) through (2d) in all m iterations in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) is used to recognize and to label independent sets among 2^n possible choices. We use the Boolean formula $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ in equation (5-2) to represent the straightforward Boolean circuit for labelling legal independent sets among 2^n possible choices in Figure 4-3. We show how to implement each instruction in the flowchart of Figure 4-3 to complete the proof. In Figure 4-3, in statement S_1, the value of the loop index variable k is set to one (1). Next, in statement S_2 in Figure 4-3, the conditional judgement of the first loop is executed. If the value of k is less than or equal to the value of m, then the *next executed* instruction is statement S_3 in Figure 4-3. Otherwise, in statement S_6 in Figure 4-3, an *End* instruction is executed to terminate the task of recognizing legal independent sets among 2^n possible choices.

We assume that the kth edge e_k is (v_i, v_j) and bits y_i and y_j are used to respectively represent vertices v_i and v_j. Next, in statement S_3 in Figure 4-3, the choices that include one vertex (v_i or v_j) or zero vertices are labeled and the choices that include two vertices v_i and v_j are discarded. This is to say that the legal independent sets satisfy the formula of the form $(\overline{y_\iota \wedge y_J})$. Hence, one **CCNOT** gate, $(|l_k^1 \oplus y_i \bullet y_j\rangle)$, with the target bit l_k^1 and the two controlled bits y_i and y_j are used to implement a ***NAND*** gate "$l_k \leftarrow \overline{y_\iota \wedge y_J}$" in statement S_3 of Figure 4-3 and the result of implementing $(\overline{y_\iota \wedge y_J})$ is written into the target bit $|l_k^1\rangle$. Next, in statement S_4 in Figure 4-3, one **CCNOT** gate, $(|o_k^0 \oplus l_k \bullet o_{k-1}\rangle)$, with the target bit o_k^0 and the two controlled bits l_k and o_{k-1} are used to implement a logical and operation "$o_k \leftarrow l_k \wedge o_{k-1}$" that is the kth ***AND*** gate in $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ in equation (5-2). The result of implementing $(l_k \wedge o_{k-1})$ is written into the target bit $|o_k^0\rangle$.

Next, in statement S_5 in Figure 4-3, the value of the index variable k in the first loop is incremented. Repeat to execute statements S_2 through S_5 in Figure 4-3 until in S_2 the conditional judgement attains a *false* value. Based on Figure 4-3, the total number of ***NAND*** gates is m. The total number of logical and operation uses m ***AND*** gates. Therefore, the cost of the quantum gate for recognizing the labelling of legal independent sets among 2^n possible choices is $(2 \times m)$ ***CCNOT*** gates. As shown in the proof of **Lemma 4-3**, there is a ***true dependence*** between statements S_3 and S_4 in Figure 4-3. The ***true dependence*** between S_3 and S_4 cannot be broken. Therefore, each statement in Figure 4-3 must be executed in sequential mode. Based on the statements above, the quantum circuit **LIS** in Figure 5-2 can implement the straightforward Boolean circuit $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ in equation (5-2).

Each bit in $|l_m^1\ l_{m-1}^1 \ldots l_1^1\rangle$ in Figure 5-2 is an auxiliary quantum bit and is used to store the result of evaluating each clause of the form $(\overline{y_\iota \wedge y_J})$. Hence, this step requires m **NAND** operations through the relations $(|l_k^1 \oplus y_i \bullet y_j\rangle)$ with the target bit l_k^1 and the two control bits y_i and y_j where $1 \le i$ and $j \le n$ and $1 \le k \le m$. In Figure 5-2, each bit in $|o_m^0\ o_{m-1}^0 \ldots o_1^0\ o_0^1\rangle$ is also an auxiliary quantum bit, and it is used to store the result of evaluating the $(k-1)$th clause and the kth clause where $1 \le k \le m$. This step requires m **AND** operations through the relation $(|o_k^0 \oplus l_k \bullet o_{k-1}\rangle)$ with the target bit o_k^0 and the two control bits l_k and o_{k-1} for $1 \le k \le m$. From the statements above, it is at once inferred that the quantum circuit, **LIS**, in Figure 5-2 with $(2 \times m)$ ***CCNOT*** gates can implement the straightforward Boolean circuit $(\wedge_{k=1}^{m}(\overline{y_\iota \wedge y_J}))$ in equation (5-2) and is the *best* quantum circuit *known* for labelling legal independent sets among 2^n possible choices. ■

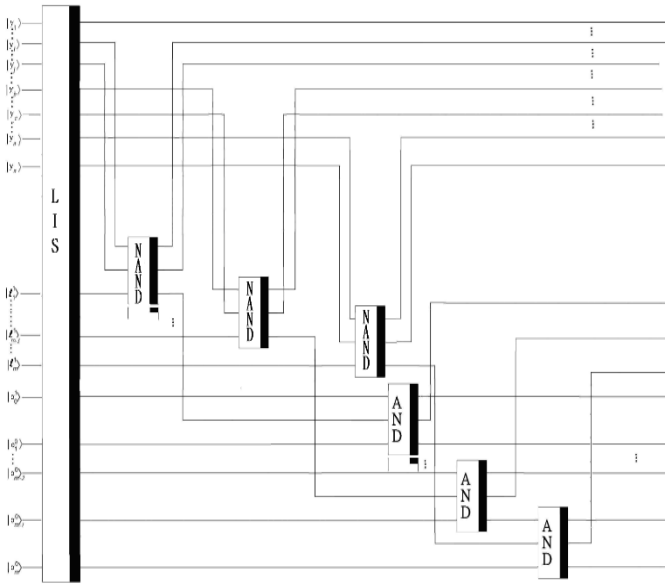

Fig. 5-2: The quantum circuit, LIS, used to label legal independent sets among 2^n possible choices.

E. Quantum Circuits and Mathematical Solutions of Molecular Solutions to the Maximum-sized Independent Sets

The straightforward Boolean circuits in Figure 4-4 obtained from Steps (4a) through (4b) at each iteration in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m) count the number of vertices in each solution. The straightforward Boolean circuits in Figure 4-4 for counting the number of vertices in each legal independent sets are

$(w_{1,1} \leftarrow o_m \wedge y_1)$ and $(w_{1,0} \leftarrow o_m \wedge \overline{y_1})$ and (5-3)

$(w_{i+1,j+1} \leftarrow y_{i+1} \wedge w_{i,j})$ and $(w_{i+1,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,j})$ for $1 \le i \le n-1$ and $0 \le j \le i$. (5-4)

Auxiliary quantum bits $|w_{i+1,j}\rangle$ and $|w_{i+1,i+1}\rangle$, where $0 \le i \le n-1$ and $0 \le j \le i$, are needed to execute these operations. For $0 \le i \le n-1$ and $0 \le j \le i$, each quantum bit in $|w_{i+1,j}\rangle$ and $|w_{i+1,i+1}\rangle$ is initially prepared in state $|0\rangle$. We assume that for $0 \le i \le n-1$ and $0 \le j \le i$, quantum bit $|w_{i+1,j+1}\rangle$ will record the status of tube (set) Y_{j+1} that has $(j+1)$ ones after the influence of y_{i+1} on the number of ones. We also suppose that for $0 \le i \le n-1$ and $0 \le j \le i$, quantum bit $|w_{i+1,j}\rangle$ is to record the status of tube (set) Y_j that has j ones after the influence of y_{i+1} to the number of ones. We use **Lemma 5-3** to show how the quantum circuits from Figures 5-3 through 5-4 implement the

straightforward Boolean circuit in equation (5-3) and equation (5-4) for counting the number of vertices in each legal independent-set.

Lemma 5-3: The quantum circuit **CFFV** in Figure 5-3 implements the straightforward Boolean circuit ($w_{1,1} \leftarrow o_m \wedge y_1$) and ($w_{1,0} \leftarrow o_m \wedge \overline{y_1}$) in equation (5-3). The quantum circuit **CMO** in Figure 5-4 implements the straightforward Boolean circuit ($w_{i+1,\,j+1} \leftarrow y_{i+1} \wedge w_{i,\,j}$) and ($w_{i+1,\,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,\,j}$) for $1 \le i \le n-1$ and $0 \le j \le i$ in equation (5.4).

Proof:

If a legal independent set has the first vertex v_1, then it satisfies the straightforward Boolean circuit ($w_{1,1} \leftarrow o_m \wedge y_1$) in equation (5-3). Otherwise, it satisfies the straightforward Boolean circuit ($w_{1,0} \leftarrow o_m \wedge \overline{y_1}$) in equation (5-3). After the influence of y_1 on the number of ones is determined, quantum bit $|w_{1,\,1}\rangle$ that encodes bit $w_{1,1}$ will record which legal independent sets have only one value 1 and contain the first vertex v_1. Quantum bit $|w_{1,\,0}\rangle$ that encodes bit $w_{1,0}$ will record which legal independent sets have *zero* ones and do not contain the first vertex v_1. Therefore, one **CCNOT** gate ($|w_{1,\,1}{}^0 \oplus o_m \bullet y_1\rangle$) with the target bit $|w_{1,\,1}{}^0\rangle$ and two control bits $|o_m\rangle$ and $|y_1\rangle$ implements ($w_{1,1} \leftarrow o_m \wedge y_1$) (the first condition of equation (5-3)). One **NOT** gate operating on $|y_1\rangle$ ($|\overline{y_1}\rangle$) and another **CCNOT** gate ($|w_{1,\,0}{}^0 \oplus o_m \bullet \overline{y_1}\rangle$) with the target bit $|w_{1,\,0}{}^0\rangle$ and two control bits $|o_m\rangle$ and $|\overline{y_1}\rangle$ implement ($w_{1,0} \leftarrow o_m \wedge \overline{y_1}$) (the second condition of equation (5-3)). Next, another **NOT** gate operating on $|\overline{y_1}\rangle$ ($|y_1\rangle$) will restore $|y_1\rangle$ in $|y_n \ldots y_1\rangle$ to its superposition state. This is to say that if the value of quantum bit $|w_{1,\,1}\rangle$ is equal to one, then quantum bit $|w_{1,\,1}{}^1\rangle$ indicates which legal independent sets have only one value 1 and contain the first vertex v_1. Similarly, if the value of quantum bit $|w_{1,\,0}\rangle$ is equal to one, then quantum bit $|w_{1,\,0}{}^1\rangle$ indicates which legal independent sets do not contain the first vertex v_1 and have zero ones. In light of the statements above, the quantum circuit **CFFV** in Figure 5-3 implements the first and second conditions of equation (5-3).

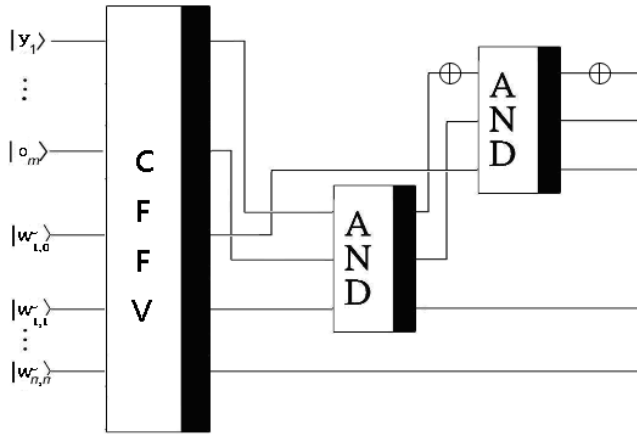

Fig. 5-3: Implementation of the first and the second conditions of equation (5-3) using the quantum circuit CFFV.

Next, if a legal independent set contains the $(i+1)$th vertex v_{i+1} and has j ones, then it satisfies the straightforward Boolean circuit ($w_{i+1,\,j+1} \leftarrow y_{i+1} \wedge w_{i,\,j}$) with $1 \le i \le n-1$ and $0 \le j \le i$ in equation (5-4). If a legal independent set does not contain the $(i+1)$th vertex v_{i+1} and has j ones, then it satisfies the straightforward Boolean circuit ($w_{i+1,\,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,\,j}$) with $1 \le i \le n-1$ and $0 \le j \le i$ in equation (5.4). After the influence of y_{i+1} on the number of ones is determined, quantum bit $|w_{i+1,\,j+1}\rangle$ that encodes bit $w_{i+1,\,j+1}$ will record which legal independent sets have $(j+1)$ ones and contain the $(i+1)$th vertex v_{i+1}. Quantum bit $|w_{i+1,\,j}\rangle$ encoding bit $w_{i+1,\,j}$ will record which legal independent sets have j ones and do not contain the $(i+1)$th vertex v_{i+1}.

Hence, one **CCNOT** gate ($|w_{i+1,\,j+1}{}^0 \oplus w_{i,\,j} \bullet y_{i+1}\rangle$) with the target bit $|w_{i+1,\,j+1}{}^0\rangle$ and two control bits $|w_{i,\,j}\rangle$ and $|y_{i+1}\rangle$ implement ($w_{i+1,\,j+1} \leftarrow y_{i+1} \wedge w_{i,\,j}$) for $1 \le i \le n-1$ and $0 \le j \le i$ (the first condition of equation (5-4)). One **NOT** gate operating on quantum bit $|y_{i+1}\rangle$ ($|\overline{y_{i+1}}\rangle$) and another **CCNOT** gate ($|w_{i+1,\,j}{}^0 \oplus w_{i,\,j} \bullet \overline{y_{i+1}}\rangle$) with the target bit $|w_{i+1,\,j}{}^0\rangle$ and two control bits $|w_{i,\,j}\rangle$ and $|\overline{y_{i+1}}\rangle$ implement ($w_{i+1,\,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,\,j}$) for $1 \le i \le n-1$ and $0 \le j \le i$ (the second condition of equation (5-4)). Next, using another **NOT** gate operating on quantum bit $|\overline{y_{i+1}}\rangle$ ($|y_{i+1}\rangle$) will restore $|y_{i+1}\rangle$ in $|y_n \ldots y_1\rangle$ to its superposition state. This implies that if the value of quantum bit $|w_{i+1,\,j+1}\rangle$ is equal to one, then quantum bit $|w_{i+1,\,j+1}{}^1\rangle$ will indicate which legal independent sets have $(j+1)$ ones and contain the $(i+1)$th vertex v_{i+1}. Similarly, if the value of quantum bit $|w_{i+1,\,j}\rangle$ is equal to one, then quantum bit $|w_{i+1,\,j}{}^1\rangle$ will indicate which legal independent sets do not contain the $(i+1)$th vertex v_{i+1} and have j ones. According to the statements above, the quantum circuit, **CMO**, in Figure 5-4 implements the first and second conditions of equation (5-4).

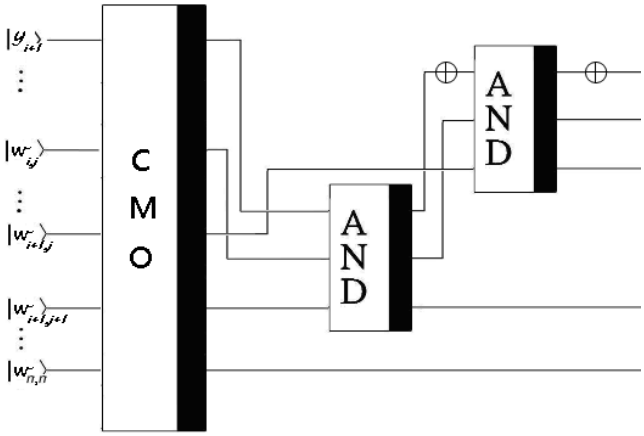

Fig. 5-4: Implementation of the first and the second conditions of (5-4) using the quantum circuit CMO.

Therefore, from the statements above can be inferred that the quantum circuit **CFFV** in Figure 5-3 can implement the straightforward Boolean circuit ($w_{1,1} \leftarrow o_m \wedge y_1$) and ($w_{1,0} \leftarrow o_m \wedge \overline{y_1}$) in equation (5-3). Similarly, it is inferred that the quantum circuit **CMO** in Figure 5-4 can implement the straightforward Boolean circuit ($w_{i+1,\,j+1} \leftarrow y_{i+1} \wedge w_{i,\,j}$) and ($w_{i+1,\,j} \leftarrow \overline{y_{i+1}} \wedge w_{i,\,j}$) for $1 \le i \le n-1$ and $0 \le j \le i$ in equation (5.4). ■

F. Quantum Circuits and Mathematical Solutions for Reading Molecular Solutions for the Maximum-sized Independent Sets

The 2^n possible molecular solutions that are created by Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m) are initialized in the

distribution: ($\frac{1}{\sqrt{2^n}}\frac{1}{\sqrt{2^n}}\frac{1}{\sqrt{2^n}}\cdots\frac{1}{\sqrt{2^n}}$). This indicates that there is the same amplitude in each of the 2^n possible molecular solutions. The previously proposed quantum circuits have labelled the answer(s), but the amplitude or probability of finding the answer(s) will decrease exponentially. Hence, based on [26], the diffusion operator is applied to increase exponentially the amplitude or probability of finding the answer(s), and is defined by matrix G as follows: $G_{i,j} = (2 / 2^n)$ if $i \neq j$ and $G_{i,i} = (-1 + (2 / 2^n))$. **Algorithm 5-1** is used to measure the answer(s) that are generated by Steps (5a) and (5b) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*).

For convenience of presentation, we assume that $|y_b{}^1\rangle$, $|l_k{}^1\rangle$, $|o_k{}^1\rangle$, $|w_{i+1,j}{}^1\rangle$ and $|w_{i+1,i+1}{}^1\rangle$ for $1 \le b \le n$, $0 \le k \le m$, $0 \le i \le n-1$, and $0 \le j \le i$, subsequently, represent the fact that the value of their corresponding quantum bits is 1. We further assume that $|y_b{}^0\rangle$, $|l_k{}^0\rangle$, $|o_k{}^0\rangle$, $|w_{i+1,j}{}^0\rangle$ and $|w_{i+1,i+1}{}^0\rangle$ for $1 \le b \le n$, $0 \le k \le m$, $0 \le i \le n-1$, and $0 \le j \le i$, subsequently, represent tha fact that the value of their corresponding quantum bits is 0. Furthermore, we have made use of the notation from **Algorithm 5-1** below in previous subsections. We use the first parameter *t* in **Algorithm 5-1** to represent the maximum size of vertex sets among legal answers, and the execution of Step (1a) in **Algorithm 5-2** in the next subsection passes its value.

Algorithm 5-1 (*t*): Mathematical solutions obtained by reading molecular solutions of the maximum-sized independent sets for any graph *G* with *m* edges and *n* vertices.

(0) A unitary operator, $\mathbf{U}_{init} = (H)\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0} I_{2\times 2})\ (\bigotimes_{k=m}^{1} I_{2\times 2})\ (I_{2\times 2})\ (\bigotimes_{k=m}^{1} I_{2\times 2})\ (H^{\otimes n})$, operates on an initial quantum state vector, $(|1\rangle)\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}{}^0\rangle)\ (\bigotimes_{k=m}^{1}|o_k{}^0\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k{}^1\rangle)\ (\bigotimes_{b=n}^{1}|y_b{}^0\rangle)$, and the 2^n possible choices of *n* bits (containing all possible independent sets) are

$$|\varphi_{0,0}\rangle = (\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle))\ \tfrac{1}{\sqrt{2^n}}\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}{}^0\rangle)\ (\bigotimes_{k=m}^{1}|o_k{}^0\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k{}^1\rangle)\ (\bigotimes_{b=n}^{1}(|y_b{}^0\rangle + |y_b{}^1\rangle)).$$

(1) For labeling which among the 2^n possible choices are legal independent sets and which are not answers, a quantum circuit in Figure 5-2, $(I_{2\times 2})\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0} I_{2\times 2})$ **(LIS)**, is used to operate on the quantum state vector $|\varphi_{0,0}\rangle$, and the following new quantum state vector is obtained

$$|\varphi_{1,0}\rangle = (\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle))\ \tfrac{1}{\sqrt{2^n}}\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}{}^0\rangle)\ (\sum_{y=0}^{2^n-1}(\bigotimes_{k=m}^{1}|o_k{}^0 \oplus l_k \bullet o_{k-1}\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k{}^1 \oplus y_i \bullet y_j\rangle)\ (|y\rangle)).$$

(2) For implementing $(w_{1,1} \leftarrow o_m \wedge y_1)$ and $(w_{1,0} \leftarrow o_m \wedge \overline{y_1})$ in equation (5-3), a quantum circuit in Figure 5-3, $(I_{2\times 2})$ **(CFFV)**, is applied to the quantum state vector $|\varphi_{1,0}\rangle$, and the following new quantum state vector is

$$|\varphi_{2,0}\rangle = (\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle))\ \tfrac{1}{\sqrt{2^n}}\ (\bigotimes_{i=n}^{2}\bigotimes_{j=i}^{0}|w_{i,j}{}^0\rangle)\ (\sum_{y=0}^{2^n-1}(|w_{1,1}{}^0 \oplus o_m \bullet y_1\rangle)\ (|w_{1,0}{}^0 \oplus o_m \bullet \overline{y_1}\rangle)\ (\bigotimes_{k=m}^{1}|o_k\rangle)\ (|o_0{}^1\rangle)(\bigotimes_{k=m}^{1}|l_k\rangle)\ (|y\rangle)).$$

(3) **For** $i = 1$ **to** $n-1$

(4) **For** $j = i$ **down to** 0

(4a) A quantum circuit in Figure 5-4, $(I_{2\times 2})$ **(CMO)**, is to determine the number of vertices among the legal independent sets and operates on the quantum state vector $(|\varphi_{2+\left(\sum_{\theta_1=0}^{i-1}(\theta_1+1)\right)+(i-j),0}\rangle)$. Since Step (4a) is embedded in the only loop, after repeateadly executing the quantum circuit in Figure 5-4, $(I_{2\times 2})$ **(CMO)**, the resulting state vector for calculating the number of vertices in each legal independent set is

$$|\varphi_{2+\frac{n^2+n-2}{2},0}\rangle = (\tfrac{1}{\sqrt{2}}(|0\rangle-|1\rangle))\ \tfrac{1}{\sqrt{2^n}}(\sum_{y=0}^{2^n-1}(\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}\rangle)\ (\bigotimes_{k=m}^{1}|o_k\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k\rangle)\ (|y\rangle)).$$

End For

End For

(5) A ***CNOT*** gate $(\frac{|0\rangle-|1\rangle}{\sqrt{2}} \oplus w_{n,t})$ with the target bit $|\frac{|0\rangle-|1\rangle}{\sqrt{2}}\rangle$ and the control bit $|w_{n,t}\rangle$ labels the legal independent set(s) with the maximum number of vertices in the quantum state vector $(|\varphi_{2+\frac{n^2+n-2}{2},0}\rangle)$, and the following new quantum state vector is

$$|\varphi_{2+\frac{n^2+n-2}{2}+1,0}\rangle = (\tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle))\ \tfrac{1}{\sqrt{2^n}} \times (-1)^{w_{n,t}}\ (\sum_{y=0}^{2^n-1}(\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}\rangle)\ (\bigotimes_{k=m}^{1}|o_k\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k\rangle)\ (|y\rangle)).$$

(6) Because quantum operations are reversible by nature, reversing all the operations carried out by Steps (4a), (2) and (1) can restore the auxiliary quantum bits to their initial states.

(7) Apply the diffusion operator to the quantum state vector produced in Step (6).

(8) Repeatedly execute Step (1) to Step (7) at most $O(\sqrt{\frac{2^n}{R}})$ times, where the value of *R* is the number of solutions and can be efficiently determined with the quantum counting algorithm [28, 41].

(9) The answer is obtained with a probability of success of at least (1 / 2) after a measurement is completed.

End Algorithm

Lemma 5-4: The output of **Algorithm 5-1** is mathematical solutions obtained by reading molecular solutions of the maximum-sized independent sets for any graph *G* with *m* edges and *n* vertices.

Proof:

Since there are 2^n possible choices (including all possible independent sets) to the independent set problem for any graph *G* with *m* edges and *n* vertices, a quantum register of *n* bits $(\bigotimes_{b=n}^{1}|y_b\rangle)$ can represent 2^n choices with initial state vector $(\bigotimes_{b=n}^{1}|y_b{}^0\rangle)$. The independent set problem for any graph *G* with *m* edges and *n* vertices requires finding a maximum-sized independent set in *G*, so those auxiliary quantum registers are necessary. By executing Step (0), an initial state vector $|\Omega\rangle = (|1\rangle)\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0}|w_{i,j}{}^0\rangle)\ (\bigotimes_{k=m}^{1}|o_k{}^0\rangle)\ (|o_0{}^1\rangle)\ (\bigotimes_{k=m}^{1}|l_k{}^1\rangle)$

($\bigotimes_{b=n}^{1}|y_b{}^0\rangle$) starts the quantum computation of the independent set problem. A unitary operator, $\mathbf{U}_{init} = (H)\ (\bigotimes_{i=n}^{1}\bigotimes_{j=i}^{0} I_{2\times 2})\ (\bigotimes_{k=m}^{1} I_{2\times 2})\ (I_{2\times 2})\ (\bigotimes_{k=m}^{1} I_{2\times 2})\ (H^{\otimes n})$, operates on the initial state vector $|\Omega\rangle$, and the resulting state vector becomes $|\varphi_{0,0}\rangle$ with 2^n choices. This indicates that 2^n possible molecular choices generated by Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) can be implemented by Step (0) in **Algorithm 5-1**.

Next, Step (1) in **Algorithm 5-1** acts as the unitary operator **LIS** which is the quantum circuit in Figure 5-2. On the execution of Step (1) in **Algorithm 5-1**, those choices among the 2^n possible are labeled that satisfy the *straightforward* Boolean circuit ($\bigwedge_{k=1}^{m}(\overline{y_\iota \wedge y_j})$) in equation (5-2). After the execution of Step (1) has been completed, the resulting state vector $|\varphi_{1,0}\rangle$ is obtained, containing those choices with $|o_m{}^1\rangle$ that indicate them to be legal independent sets and those illegal choices with $|o_m{}^0\rangle$ that do not satisfy the condition. Hence, the straightforward Boolean circuit ($\bigwedge_{k=1}^{m}(\overline{y_\iota \wedge y_j})$) in equation (5-2) generated by Steps (2a) through (2d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, *n, m*) can be implemented by Step (1) in **Algorithm 5-1**.

Next, Step (2) in **Algorithm 5-1** acts as the unitary operator **CFFV** that corresponds to the quantum circuit in Figure 5-3. On the execution of Step (2) in **Algorithm 5-1**, the number of ones from the influence of the first vertex in each legal independent set is computed. After the execution of Step (2), the state vector $|\varphi_{2,0}\rangle$ is obtained, which includes those legal independent sets with $|w_{1,1}{}^1\rangle$ that have *one* ones and contain the *first* vertex and those legal independent sets with $|w_{1,0}{}^1\rangle$ that have *zero* ones and do not contain the first vertex. This implies that the straightforward Boolean circuit ($w_{1,1} \leftarrow o_m \wedge y_1$) and ($w_{1,0} \leftarrow o_m \wedge \overline{y_1}$) in equation (5-3) generated by Steps (4a) and (4b) in the *first* iteration (0, 0) in **Solve-independent-set-problem**(Y_0, *n, m*) can be implemented by Step (2) in **Algorithm 5-1**.

Next, Step (4a) is the only statement in the first loop in **Algorithm 5-1** and works as the unitary operator **CMO** which corresponds to the quantum circuit in Figure 5-4. This step is to determine the number of ones (the number of vertices) among the legal independent sets. Step (3) and Step (4) consist each of a two-level loop. When the value of the index variable *i* is equal to one and the value of the index variable *j* is from one down to zero, Step (4a) is executed repeatedly two times. Similarly, when the value of the index variable *i* is equal to two and the value of the index variable *j* is from two down to zero, Step (4a) is executed repeatedly three times. Similarly, when the value of the index variable *i* is equal to ($n - 1$) and the value of the index variable *j* is from ($n - 1$) down to zero, Step (4a) is repeatedly executed *n* times. This is to say that the total number of executions of Step (4a) is $(2 + 3 + \ldots n) = (n^2 + n - 2) / 2$. Because the state vector $|\varphi_{2,0}\rangle$ is generated from Step (2) and its index is 2 (two), after repeatedly executing Step (4a), we use $2 + ((n^2 + n - 2) / 2)$ as the index of the resulting state and the resulting state vector $|\varphi_{2+\frac{n^2+n-2}{2},0}\rangle$ is obtained in which the number of vertices in each legal independent set is calculated. This indicates that the straightforward Boolean circuit ($w_{i+1,\,j+1} \leftarrow y_{i+1} \wedge w_{i,\,j}$) and ($w_{i+1,\,j} \leftarrow \overline{y_{\iota+1}} \wedge w_{i,\,j}$) for $1 \le i \le n - 1$ and $0 \le j \le i$ in equation (5-4) generated in Steps (4a) and (4b) in the same iteration (*i*, *j*) in **Solve-independent-set-problem**(Y_0, *n, m*) can be implemented by Step (4a) in **Algorithm 5-1**.

Next, one **CNOT** gate, ($\frac{|0\rangle-|1\rangle}{\sqrt{2}} \oplus w_{n,\,t}$) with the target bit $|\frac{|0\rangle-|1\rangle}{\sqrt{2}}\rangle$ and the control bit $|w_{n,\,t}\rangle$, in the execution of Step (5) of **Algorithm 5.1** labels the answer(s) with the phase (−1). The resulting state vector $|\varphi_{2+\frac{n^2+n-2}{2}+1,0}\rangle$ consists of the part of the answer with the phase (−1) and the other part with the phase (+1). Because quantum operations are reversible by nature, the execution of Step (6) will reverse all these operations completed by Step (4a), Step (2) and Step (1) that can restore the auxiliary quantum bits to their initial states. Next, on the execution of Step (7) in **Algorithm 5-1**, the diffusion operator is applied to complete the task of increasing the probability of success in measuring the answer(s). In Step (8) in **Algorithm 5-1**, after repeatedly executing Steps (1) through (7) $O(\sqrt{\frac{2^n}{R}})$ times, a maximum probability of success is generated. Next, by executing Step (9) in **Algorithm 5-1**, a measurement is obtained and the answer(s) is/are returned to **Algorithm 5-2**. Because the result produced by each step in **Algorithm 5-1** is a unit vector in a finite-dimensional Hilbert space, therefore, we at once infer that the output of **Algorithm 5-1** is mathematical solutions obtained by reading molecular solutions of the maximum-sized independent sets to any graph *G* with *m* edges and *n* vertices. ■

G. Solving the Independent Set Problem on any Graph G with m Edges and n Vertices

The following algorithm solves the independent-set problem for any graph *G* with *m* edges and *n* vertices. We have used the notations used in **Algorithm 5-2** in the previous subsections.

Algorithm 5-2: Solving the independent set problem for any Graph G with *m* edges and *n* vertices.

(1) **For** $t = n$ **to** 1

(1a) Call **Algorithm 5-1**(*t*).

(1b) **If** the answer is obtained from the *t*th execution of Step

(1a) **then**

(1c) Terminate **Algorithm 5-2**.

End If

End For

End Algorithm

Lemma 5-5: **Algorithm 5-2** obtains the maximum-sized independent sets to the independent set problem in any graph *G* with *m* edges and *n* vertices.

Proof:

In each execution of Step (1a) in **Algorithm 5-2**, **Algorithm 5-1** is called to complete two main tasks. The first task is to calculate the number of vertices in each legal independent set. This demonstrates that mathematical solutions of molecular solutions for finding the maximum-sized independent sets in the independent set problem for any graph *G* with *m* edges and *n* vertices are a unit vector in the finite-dimensional Hilbert space. The second task is to use the

diffusion operator that increases exponentially the probability of success in measuring the answer(s) from molecular solutions of the maximum-sized independent sets. By this we show that mathematical solutions obtained by reading molecular solutions of the maximum-sized independent sets for any graph G with m edges and n vertices are still a unit vector in the finite-dimensional Hilbert space. Next, in each execution of Step (1b) in **Algorithm 5-2,** if from the tth execution of Step (1a) in **Algorithm 5-2** the answer(s) is(are) found, then the tth execution of Step (1c) in **Algorithm 5-2** will terminate **Algorithm 5-2**. Otherwise, repeatedly execute Steps (1a) through (1c) until the answer(s) to the independent set problem for any graph G with m edges and n vertices is(are) found. Hence, it is immediately derived that **Algorithm 5-2** can be used to obtain the answer(s) to the independent set problem for any graph G with m edges and n vertices. ■

H. Durr-Hoyer's Algorithm and Ahuja-Kapoor's Algorithm and the Quantum Existence Testing not Solving the Independent Set Problem for any Graph G with m Edges and n Vertices

Many information processing and computing problems can be traced back to the problem of finding an extremum of a database or a cost function. The Durr–Hoyer algorithm [48] finds the minimum value(s) satisfying any given condition in an unsorted database or a cost function with 2^n items. Ahuja–Kapoor's algorithm [49] finds the maximum value(s) satisfying any given condition in an unsorted database or a cost function with 2^n items. In order to improve the performance of the two algorithms, quantum existence testing which integrates quantum counting and binary search [31] can be used to find the minimum value(s) or the maximum values satisfying any given condition in an unsorted database or a cost function with 2^n items. The independent set problem for any graph G with m edges and n vertices entails finding the maximum-sized independent set with the maximum number of vertices in an unsorted database or a cost function with 2^n subsets of vertices. We use the following lemma to show why Durr–Hoyer's algorithm, Ahuja–Kapoor's algorithm and quantum existence testing cannot solve the independent set problem for any graph G with m edges and n vertices.

Lemma 5-6: Durr–Hoyer's algorithm, Ahuja–Kapoor's algorithm and quantum existence testing cannot solve the independent set problem for any graph G with m edges and n vertices.

Proof:

In Durr–Hoyer's algorithm, Ahuja–Kapoor's algorithm and in quantum existence testing algorithm, the solution space Y is a set of 2^n possible choices and Y is equal to $\{y_n\, y_{n-1} \ldots y_2\, y_1 | \forall\, y_d \in \{0, 1\}$ for $1 \le d \le n\}$. This indicates that the length of each element in Y is n bits and that each element represents one of the 2^n possible choices. For the sake of presentation, we assume that $y_d{}^0$ indicated that the value of y_d is zero and $y_d{}^1$ indicates that the value of y_d is one. The *first* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^0$ encodes the decimal value 0 (zero). The *second* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^0\, y_1{}^1$ encodes the decimal value 1 (one). The *third* element $y_n{}^0\, y_{n-1}{}^0 \ldots y_2{}^1\, y_1{}^0$ encodes the decimal value 2 (two). And so on, with the *last* element $y_n{}^1\, y_{n-1}{}^1 \ldots y_2{}^1\, y_1{}^1$ encoding the decimal value $2^n - 1$. Because the solution space does not contain any subsets of vertices, these three algorithms cannot find the maximum-sized independent set(s). Therefore, from the statements above, we at once derive that Durr–Hoyer's algorithm, Ahuja–Kapoor's algorithm and quantum existing testing cannot solve the independent set problem for any graph G with m edges and n vertices. ■

VI. Complexity Assessment

In this section, we estimate the time complexity and the spatial complexity of **Algorithm 5-2** for solving the independent set problem for any graph G with m edges and n vertices. Subsequently, we demonstrate that **Algorithm 5-2** provides a quadratic speedup for solving the independent set problem for any graph G with m edges and n vertices, which is the *best* speed-up *known* for the problem.

A. The Time and Space Complexity of Algorithm 5-2

Lemma 6-1: The best case time complexity for **Algorithm 5-2** involves $((2^{n/2} \times (2 \times n)) + (n + 1))$ **Hadamard** gates, $(2^{n/2} \times (2 \times (n^2 + n)))$ **NOT** gates, $(2^{n/2})$ **CNOT** gates, $(2^{n/2} \times (4 \times m + (2 \times (n^2 + n))))$ **CCNOT** gates, $(2^{n/2})$ phase shift gates of n quantum bits and a quantum measurement.

Proof:

In **Algorithm 5-2**, Step (1) is the main loop and the steps embedded in this mail loop are executed in n iterations. Hence, the first execution of Step (1a) invokes **Algorithm 5-1**. In Step (0) of **Algorithm 5-1**, $(n + 1)$ **Hadamard** gates are applied. Next, in Step (1) of **Algorithm 5-1**, $(2 \times m)$ **CCNOT** gates are applied. Next, in Step (2) of **Algorithm 5-1**, two **CCNOT** gates and two **NOT** gates are applied. Then, Step (4a) of **Algorithm 5-1** is the only instruction in the first loop, and Step (4a) of **Algorithm 5-1** results in $(n^2 + n - 2)$ **NOT** gates and $(n^2 + n - 2)$ **CCNOT** gates. Next, in Step (5) of **Algorithm 5-1**, one **CNOT** gate is applied. Then, Step (6) of **Algorithm 5-1** restores the auxiliary quantum bits back to their original status. Therefore, Step (6) of **Algorithm 5-1** generates $(n^2 + n)$ **NOT** gates and $((2 \times m) + (n^2 + n))$ **CCNOT** gates. This is to say that Steps (1) through (6) complete the oracle work and label the answer(s) with the phase (−1). It is very clear from Step (7) of **Algorithm 5-1** that one diffusion operator is executed. In Step (8) of **Algorithm 5-1**, implementing $(\sqrt{2^n})$ oracle works and $(\sqrt{2^n})$ diffusion operators is the worst case because the value of R is equal to one and this case is the worst case. We suppose that a phase shift gate $\boldsymbol{U_{PSG}}$ of n quantum bits acts as follows:

$$\boldsymbol{U_{PSG}}: \begin{cases} |x> \rightarrow -|x>, & x \neq 0 \\ |0> \rightarrow |0> \end{cases}.$$

Since according to [28, 41] a single physical operation can accomplish the controlled phase shift gate of n quantum bits $\boldsymbol{U_{PSG}}$, it to a fundamental gate. From [28, 41] we have that the decomposition of the diffusion operator, $H^{\otimes n}\, \boldsymbol{U_{PSG}}\, H^{\otimes n}$, can implement each diffusion operator. Next, in Step (9) of **Algorithm 5-1**, a measurement is carried out. Therefore, after the first call of **Algorithm 5-1**, it is derived that $((2^{n/2} \times (2 \times n)) + (n + 1))$ **Hadamard** gates, $(2^{n/2} \times (2 \times (n^2 + n)))$ **NOT** gates, $(2^{n/2})$ **CNOT** gates, $(2^{n/2} \times (4 \times m + (2 \times (n^2 + n))))$ **CCNOT** gates, $(2^{n/2})$ phase shift gates of n quantum bits and a quantum measurement are implemented.

After the first call of **Algorithm 5-1** has been completed, and then in the first execution of Step (1b) in **Algorithm 5-2**, if the first execution of Step (1a) in **Algorithm 5-2** returns the answer, then **Algorithm 5-2** is terminated in the first execution of Step (1c) in **Algorithm 5-2**. Therefore, the best case of the time complexity for **Algorithm 5-2** involves $((2^{n/2} \times (2 \times n)) + (n + 1))$ **Hadamard** gates, $(2^{n/2} \times (2 \times (n^2 + n)))$ **NOT** gates, $(2^{n/2})$ **CNOT** gates, $(2^{n/2} \times (4 \times m + (2 \times (n^2 + n))))$ **CCNOT** gates, $(2^{n/2})$ phase shift gates of n quantum bits and a quantum measurement. ■

Lemma 6-2: The worst case time complexity for **Algorithm 5-2** is $(n \times ((2^{n/2} \times (2 \times n)) + (n + 1)))$ **Hadamard** gates, $(n \times (2^{n/2} \times (2 \times (n^2 + n))))$ **NOT** gates, $(n \times 2^{n/2})$ **CNOT** gates, $(n \times (2^{n/2} \times (4 \times m + (2 \times (n^2 + n)))))$ **CCNOT** gates, $(n \times 2^{n/2})$ phase shift gates of n quantum bits and (n) quantum measurements.

Proof:

Based on **Algorithm 5-2**, for solving the independent set problem for any graph G with m edges and n vertices, the worst case is to find the answer after a measurement of the result yielded from the nth execution of Step (1a) in **Algorithm 5-2** is completed. This is to say that Step (1a) and Step (1b) in **Algorithm 5-2** are executed n times and Step (1c) in **Algorithm 5-2** is executed once. Therefore, the worst case time complexity for **Algorithm 5-2** is $(n \times ((2^{n/2} \times (2 \times n)) + (n + 1)))$ **Hadamard** gates, $(n \times (2^{n/2} \times (2 \times (n^2 + n))))$ **NOT** gates, $(n \times 2^{n/2})$ **CNOT** gates, $(n \times (2^{n/2} \times (4 \times m + (2 \times (n^2 + n)))))$ **CCNOT** gates, $(n \times 2^{n/2})$ phase shift gates of n quantum bits and (n) quantum measurements. ■

Lemma 6-3: The worst and the best case spatial complexity for solving the independent set problem for any graph G with m edges and n vertices are the same: $((2 \times m + 2 \times n + 2) + ((n \times (n + 1)) / 2))$ quantum bits.

Proof:

As for any graph G with m edges and n vertices there are 2^n possible choices (including all possible independent sets) for solving the independent set problem, using a quantum register with n quantum bits $(\bigotimes_{b=n}^{1} |y_b^0\rangle)$ encodes 2^n choices. The independent set problem for any graph G with m edges and n vertices is to find a maximum-sized independent set of G. This is possible by using the auxiliary quantum registers. The initial states of those auxiliary quantum registers are $(|1\rangle)$ $(\bigotimes_{i=n}^{1} \bigotimes_{j=i}^{0} |w_{i,j}^0\rangle)$ $(\bigotimes_{k=m}^{1} |o_k^0\rangle)$ $(|o_0^1\rangle)$ $(\bigotimes_{k=m}^{1} |l_k^1\rangle)$. Based on **Algorithm 5-2** we have than the best case spatial complexity for **Algorithm 5-2** is to find the answer after implementing **Algorithm 5-1** once. Hence, the best case spatial complexity for **Algorithm 5-2** involves $((2 \times m + 2 \times n + 2) + ((n \times (n + 1)) / 2))$ quantum bits. Since quantum bits can be reused, the worst case is still $((2 \times m + 2 \times n + 2) + ((n \times (n + 1)) / 2))$ quantum bits. Hence, it is at once inferred that the worst and the best case spatial complexity for **Algorithm 5-2** are the same, and they both are equal to $((2 \times m + 2 \times n + 2) + ((n \times (n + 1)) / 2))$ quantum bits. ■

B. Proof of a Quadratic Speedup for Solving the Independent Set Problem for any Graph G with m Edges and n Vertices

Lemma 6-4: **Algorithm 5-2** gives a quadratic speed-up for solving the independent set problem for any graph G with m edges and n vertices. This speedup is the *best* speed-up *known* for the problem.

Proof:

Bennett et al. [8] have proved that a quadratic speed-up for classical algorithms is the *best* speed-up *known* for solving any NP-complete problem. From **Lemma 6-2** we have that the worst case of **Algorithm 5-2** for solving the independent set problem of any graph G with m edges and n vertices matches a quadratic speed-up for classical algorithms. Hence, we immediately derive that **Algorithm 5-2** gives a quadratic speed-up, which is the *best* speed-up *known* for solving the independent set problem for any graph G with m edges and n vertices. ■

VII. Mathematical Representation of Molecular Solutions for Independent Set Problem for any graph with m Edges and n Vertices

We use the following lemma to demonstrate that mathematical solutions of molecular solutions for solving the independent set problem for any graph G with m edges and n vertices are a unit vector in the finite-dimensional Hilbert space.

Lemma 7-1: Mathematical solutions of molecular solutions for solving the independent set problem for any graph G with m edges and n vertices are a unit vector in the finite-dimensional Hilbert space.

Proof:

In Steps (0a) through (1d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m), 2^n possible choices (independent sets) encoded by 2^n DNA sequences are produced, and encoded by n Hadamard gates operating on n initial quantum bits in Step (0) in **Algorithm 5-1**. This is to say that mathematical solutions for the 2^n possible choices (independent sets) encoded by the 2^n DNA sequences are a vector unit in the finite-dimensional Hilbert space. Next, on each execution of Steps (2a) through (2d) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m), *legal* choices (legal independent sets) and *illegal* choices (illegal independent sets) among the 2^n possible choices encoded by the 2^n DNA sequences are decided. The same task can be completed by using the unitary operators from Step (1) in **Algorithm 5-1**. This indicates that mathematical solutions for legal choices and illegal choices among 2^n possible choices encoded by 2^n DNA sequences are still a unit vector in the finite-dimensional Hilbert space.

Next, on each execution of Steps (4a) through (4b) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m), the legal choices among the 2^n choices encoded by the 2^n DNA sequences are classified according to the number of vertices. The same task can be completed using the unitary operators from Step (2) and Step (4a) in **Algorithm 5-1**. This implies that the mathematical solutions of those legal choices classified among the 2^n choices encoded by the 2^n DNA sequences are still a unit vector in the finite-dimensional Hilbert space. Next, on each execution of Steps (5a) and (5b) in the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m), the *maximum-sized* independent sets encoded by the DNA sequences with the maximum number of vertices are read, and they are also read by carrying out a measurement after their amplitude has been exponentially amplified. This is to say that

the mathematical solutions for the *maximum-sized* independent sets encoded by the DNA sequences with the maximum number of vertices are still a unit vector in the finite-dimensional Hilbert space. Hence, from the statements above, we right away derive that mathematical solutions of molecular solutions for solving the independent set problem for any graph G with m edges and n vertices are a unit vector in the finite-dimensional Hilbert space. ■

VIII. Proof that Reduction among NP-complete Problems is useless and that each NP-complete Problem that has its own best Algorithm

We assume a collection $C = \{c_1, c_2, ..., c_m\}$ of clauses on a finite set U of variables, $\{u_1, u_2, \ldots, u_n\}$, such that $|c_x|$ is equal to 3 for $1 \le x \le m$, where $|c_x|$ is the number of variables in the xth clause. The 3-satisfiability problem (3-SAT) is to find whether there is a truth assignment for U that satisfies all of the clauses in C. The simple structure for the 3-SAT problem makes it one of the most widely used problems for other NP-complete results [7]. The **Cook–Levin** theorem, also known as **Cook's** theorem [50], states that the 3-satisfiability problem (3-SAT), which is one of the Boolean satisfiability problems, is NP-complete. That is, it is in NP, and any problem in NP can be reduced in polynomial time by a deterministic Turing machine that is a digital computer to the 3-satisfiability problem (3-SAT). An important consequence of this theorem is that if there exists a deterministic polynomial time algorithm for solving 3-satisfiability problem (3-SAT), then every NP problem can be solved by a deterministic polynomial time algorithm. We use **Lemma 8-1** to show that reduction among NP-complete problems is *useless* and that each NP-complete problem has its own, best quantum algorithm. **Lemma 8-2** shows then that the proposed quantum-molecular algorithm with a quadratic speed-up for solving the independent set problem in a graph G with n vertices and m edges is not the best or optimal quantum algorithm.

Lemma 8-1: Reduction among NP-complete problems is *useless*, and each NP-complete problem has its own, best quantum algorithm.

Proof:

We suppose that U is $\{u_1, u_2, \ldots, u_n\}$ and C is $\{c_1, c_2, ..., c_m\}$. U and C are any instance for the 3-SAT problem. [7] use a polynomial time algorithm to transform the 3-SAT problem with m clauses and n Boolean variables into the independent set problem for a graph G with $(3 \times n)$ vertices and $((3 \times m) + w)$ edges. w is the number of the pair $(u_i, \overline{u_i})$ in which u_i and $\overline{u_i}$ appear in different clauses and the value of w is less than the value of m. This indicates that if **Algorithm 5-2** and **Algorithm 5-1** are applied to solve the reduced 3-SAT problem, then the time complexity of the best case is $O(2^{(3\times n)/2})$. This implies that for solving the reduced 3-SAT problem **Algorithm 5-2** and **Algorithm 5-1** cannot give a quadratic speed-up and the process of reduction among NP-complete problems not only cannot speed up the performance of quantum algorithms but, to the contrary, slows it down. Therefore, from the statements above we at once infer that reduction among NP-complete problems is *useless* and that each NP-complete problem has its own best quantum algorithm. ■

Lemma 8-2: The proposed quantum-molecular algorithm with a quadratic speed-up for solving the independent set problem in a graph G with n vertices and m edges is not the best or optimal quantum algorithm.

Proof:

From **Lemma 6-1** through **Lemma 6-4** it follows that the lower and the upper bound of the time complexity for the proposed quantum-molecular algorithm for solving the independent set problem in a graph G with n vertices and m edges are, respectively, $\Omega(2^{n\times\frac{1}{2}})$ queries and $O(2^{n\times\frac{1}{2}})$ queries with $((2 \times m + 2 \times n + 2) + ((n \times (n + 1)) / 2))$ quantum bits. The proposed quantum-molecular algorithm satisfies the important result in [8] which says that a quadratic speed-up for solving any NP-complete problem is a *tight* lower bound. However, from the poof of **Lemma 8-1** it follows that a polynomial time algorithm can transform the 3-SAT problem with m clauses and n Boolean variables into the independent set problem for a graph G with $(3 \times n)$ vertices and $((3 \times m) + w)$ edges. This is to say that a reduction among NP-complete problems makes the size of the input of the reduced problem become larger than that of the *original* problem. This is the reason why the proposed quantum-molecular algorithm for solving the reduced problem cannot give any speed-up. It appears to be the case that the important result of [8] violates an important consequence of Cook's theorem [50]. Therefore, from the statements above, we at once infer that the proposed quantum-molecular algorithm with a quadratic speed-up for solving the independent set problem in a graph G with n vertices and m edges is not the best or optimal quantum algorithm. ■

IX. Proof of a quantum lower Bound of $\Omega(\sqrt{\frac{2^n}{2}})$ Queries for Solving the Element Distinctness Problem with an Input of n Bits

From [33] we have that the element distinctness problem with an input of n bits is to determine whether the given 2^n real numbers are distinct or not. A quantum lower bound of solving it is $\Omega(2^{n\times\frac{2}{3}})$ queries for a quantum walk algorithm [33]. The formal definition of the problem is as follows: given a function $\boldsymbol{H}: \{a|\ 0 \le a \le 2^n - 1\} \rightarrow \{b|\ 0 \le b \le 2^m - 1\}$, the *r-element distinctness* problem is to find r-distinct elements $a_1, a_2, ..., a_r \in \{a|\ 0 \le a \le 2^n - 1\}$ such that $\boldsymbol{H}(a_1) = \boldsymbol{H}(a_2) = ... = \boldsymbol{H}(a_r)$. Childs and Eisenberg in [51] extend the *r-element distinctness* problem to a much more general problem, namely the problem of finding a subset of size r that satisfies any given property. Childs and Eisenberg in [51] assume that there is a black-box function $\boldsymbol{O_F}: D \rightarrow R$, where the domain D is a finite set and the range R is also a finite set. They further assume that the domain D is equal to $\{X_1, X_2, \ldots, X_N\}$ and that $|D|$ represents the size of the domain D and that this size is equal to N, which is the problem size. They also assume a set $(D \times R)^r = \{((X_1, \boldsymbol{O_F}(X_1)), \ldots, (X_r, \boldsymbol{O_F}(X_r)))\ |\ X_k \in D$ and $\boldsymbol{O_F}(X_k) \in R\}$ and that there is a property $P \subset (D \times R)^r$. The more formal definition of the r-subset finding problem is to find some r-subset $\{X_1, X_2, \ldots, X_r\} \subset D$ such that $((X_1, \boldsymbol{O_F}(X_1)), \ldots, (X_r, \boldsymbol{O_F}(X_r))) \in P$, or reject if none exists. We use the following lemma to show that a new

quantum lower bound for solving it is $\Omega(\sqrt{\frac{2^n}{2}})$ queries.

Lemma 9-1: For solving the element distinctness problem with an input of n bits, the proposed quantum-molecular algorithm improves a quantum lower bound $\Omega(2^{n\times\frac{2}{3}})$ queries with a quantum walk algorithm to $\Omega(\sqrt{\frac{2^n}{2}})$ queries.

Proof:

We assume that $G = (V, E)$ is a graph where V is the set of vertices in G and E is the set of edges in G. We also suppose that V is $\{v_1, \ldots, v_n\}$ and E is $\{(v_a, v_b)|\ v_a$ and v_b are, respectively, elements in $V\}$. An *independent set* of graph G with n vertices and m edges is a subset $V^1 \subseteq V$ of vertices such that for all $v_a, v_b \in V^1$, the edge (v_a, v_b) is *not* in E [7, 9]. The independent set problem for graph G with n vertices and m edges is to find a *maximum-sized* independent set in G. For graph G with n vertices and m edges, the number of subsets of the set of n vertices is 2^n. We suppose that there is a black-box function, $\boldsymbol{O_F}$: $D \rightarrow R$ that computes which subsets of vertices are the independent sets with the *maximum* number of vertices. We also suppose that the domain D is equal to $\{y_n\, y_{n-1} \ldots y_2\, y_1|\ \forall\ y_d \in \{0, 1\}$ for $1 \le d \le n\}$.

We assume that r binary numbers of n bits in length, $X_1, X_2, \ldots, X_r$, are all elements in D. We also suppose that a set $(D \times R)^r = \{((X_1, \boldsymbol{O_F}(X_1)), \ldots, (X_r, \boldsymbol{O_F}(X_r)))\ |\ X_k \in D$ and $\boldsymbol{O_F}(X_k) \in R\}$. We assume that there is a property $P \subset (D \times R)^r$. We also suppose that each binary number of n bits, X_k, for $1 \le k \le r$ encodes a subset of vertices in which a black-box function, $\boldsymbol{O_F}$, can determine a *maximum-sized* independent set. This is to say that $((X_1, \boldsymbol{O_F}(X_1)), \ldots, (X_r, \boldsymbol{O_F}(X_r)))$ is the answer of the independent set problem for graph G with n vertices and m edges, and $\{X_1, X_2, \ldots, X_r\} \subset D$ such that $((X_1, \boldsymbol{O_F}(X_1)), \ldots, (X_r, \boldsymbol{O_F}(X_r))) \in P$. Therefore, the independent set problem of graph G with n vertices and m edges is a type of the element distinctness problem and is a type of the r-subset finding problem.

From **Lemma 6-1** through **Lemma 6-2**, for solving the independent set problem for graph G with n vertices and m edges, a quantum lower bound is $\Omega(\sqrt{\frac{2^n}{r}})$ queries and a quantum upper bound is $O(\sqrt{\frac{2^n}{r}})$ queries. When the value of r is equal to two, the quantum lower bound is $\Omega(2^{n\times\frac{2}{3}})$ queries with a quantum walk algorithm [30]. However, the proposed quantum-molecular algorithm for the value of r equal to two gives a quantum lower bound of $\Omega(\sqrt{\frac{2^n}{2}})$ queries and a quantum upper bound of $O(\sqrt{\frac{2^n}{2}})$ queries. Therefore, we at once infer that for solving the element distinctness problem with an input of n bits, the proposed quantum-molecular algorithm enhances a quantum lower bound of $\Omega(2^{n\times\frac{2}{3}})$ queries with a quantum walk algorithm to $\Omega(\sqrt{\frac{2^n}{2}})$ queries. ■

X. EXPERIMENTAL RESULTS OF FINDING THE MAXIMUM-SIZED INDEPNDENT SETS IN A GRAPH WITH TWO VERTICES AND ONE EDGE

In Fig. 10-1, graph G^1 consists of two vertices and an edge. All of the independent sets in G^1 are $\{v_1\}$, $\{v_2\}$ and $\{\}$, which is an empty set. The *maximum-sized* independent sets for G^1 are $\{v_1\}$ and $\{v_2\}$. We use the quantum circuit in Figure 10-2 and the quantum circuit in Figure 10-3 to respectively find the answer $\{v_1\}$ and the answer $\{v_2\}$. To that end, we use the backend *ibmqx4* with five quantum bits in **IBM**'s quantum computers to test our theory.

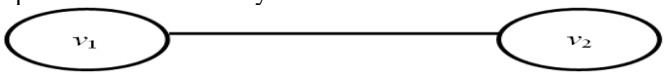

Fig. 10-1: The graph G^1 for our problem.

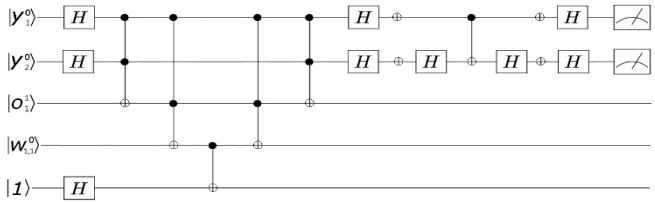

Fig. 10-2: The quantum circuit for finding the answer $\{v_1\}$.

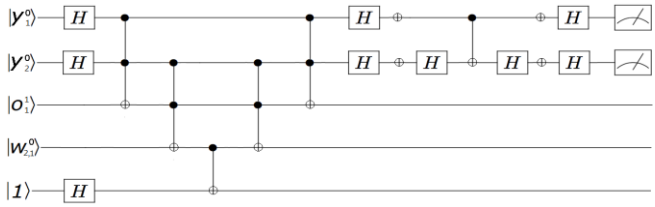

Fig. 10-3: The quantum circuit for finding the answer $\{v_2\}$.

In **IBM**'s graphical interface of *ibmqx4*, the available gates are **CNOT**, which is the only gate with two quantum bits, and other gates that act on single quantum bits. In the backend *ibmqx4* with five quantum bits, there are only six pairs of **CNOT** gates. We decompose **CCNOT** gate into six **CNOT** gates and gates of one quantum bit that appear in Figure 10-4 [30]. In Figure 10-4, H is the Hadamard gate, $T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\sqrt{-1}\times\frac{\pi}{4}} \end{bmatrix}$ and $T^+ = \begin{bmatrix} 1 & 0 \\ 0 & e^{-1\times\sqrt{-1}\times\frac{\pi}{4}} \end{bmatrix}$. In the backend *ibmqx4*, we use quantum bits q[3], q[4], q[2], q[1] and q[0] to respectively implement quantum bits $|y_1^0\rangle$, $|y_2^0\rangle$, $|o_1^1\rangle$, $|w_{1,1}{}^0\rangle$ and $|1\rangle$ in Figure 10-2. Similarly we also use quantum bits q[3], q[4], q[2], q[1] and q[0] to respectively implement quantum bits $|y_1^0\rangle$, $|y_2^0\rangle$, $|o_1^1\rangle$, $|w_{2,1}{}^0\rangle$ and $|1\rangle$ in Figure 10-3. Because in the backend *ibmqx4* a **CNOT** gate cannot be applied to quantum bits q[3] and q[1], the second and the third **CCNOT** gates in Figure 10-2 and in Figure 10-3 cannot be implemented by the backend *ibmqx4*. Therefore, we use the quantum circuit in Figure 10-5 and the quantum circuit in Figure 10-6 to respectively find the answer $\{v_1\}$ and the answer $\{v_2\}$.

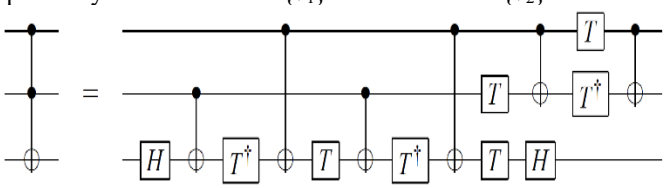

Fig. 10-4: Decomposing CCNOT gate into six CNOT gates and gates of one quantum bit.

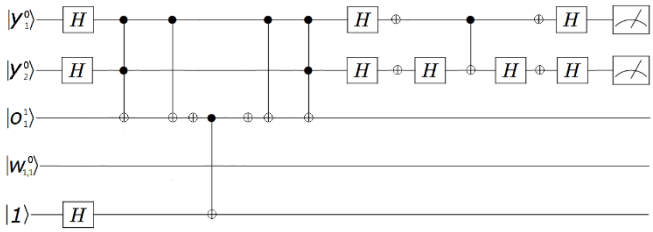

Fig. 10-5: The quantum circuit for finding the answer $\{v_1\}$ adapted for execution on the backend *ibmqx4*.

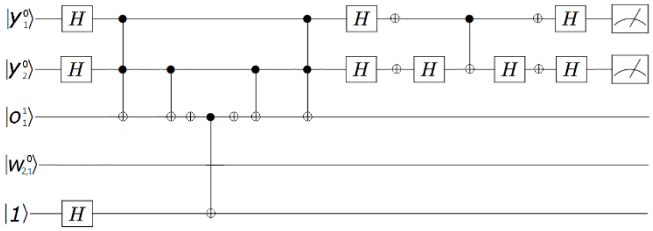

Fig. 10-6: The quantum circuit for finding the answer $\{v_2\}$ adapted for execution on the backend *ibmqx4*.

We write two programs in open quantum assembly language version 2.0 for the backend *ibmqx4* to implement the two quantum circuits in Figures 10-5 and 10-6. In the first program, we use the four statements - "OPENQASM 2.0;", "include "qelib1.inc";", "qreg q[5];" and "creg c[5];" - to declare five quantum bits with the initial state |0> and five classic bits with the initial value 0. Next, we use the two statements "x q[2];" and "x q[0];" to set q[2] and q[0] to the state |1>. Then, the three statements "h q[3];", "h q[4];", "h q[0];" are used to complete the three Hadamard gates in the first time slot in Fig. 10-5. Next, the *fifteen* statements - "h q[2];", "cx q[4],q[2];", "tdg q[2];", "cx q[3],q[2];", "t q[2];", "cx q[4],q[2];", "tdg q[2];", "cx q[3],q[2];", "t q[4];", "t q[2];", "cx q[3],q[4];", "h q[2];", "t q[3];", "tdg q[4];" and "cx q[3],q[4];" – are used to complete the first ***CCNOT*** gate in the *second* time slot of Figure 10-5. Then, the two statements "cx q[3],q[2];" and "x q[2];" implement a ***CNOT*** gate and a ***NOT*** gate for labelling the answer from the third time slot through the fourth time slot in Figure 10-5.

Next, in the first program, the statement "cx q[2],q[0];" is used to label the target state with its amplitude by (-1) in the fifth time slot in Figure 10-5. After that, seventeen statements are used to complete the reversal operations from the sixth time slot through the eighth time slot in Figure 10-5. The *seventeen* statements are "x q[2];", "cx q[3],q[2];", "cx q[3],q[4];", "tdg q[4];", "t q[3];", "h q[2];", "cx q[3],q[4];", "t q[2];", "t q[4];", "cx q[3],q[2];", "tdg q[2];", "cx q[4],q[2];", "t q[2];", "cx q[3],q[2];", "tdg q[2];", "cx q[4],q[2];" and "h q[2];".

Next, the *eleven* statements "h q[3];", "h q[4];", "x q[3];", "x q[4];", "h q[4];", "cx q[3],q[4];", "h q[4];", "x q[4];", "x q[3];", "h q[4];" and "h q[3];" are used to complete the amplification of the amplitude of the answer(s) from the *ninth* time slot through the *fifteenth* time slot in Figure 10-5. And finally, the two statements "measure q[3] -> c[3];" and "measure q[4] -> c[4];" compete the measurement of the answer(s) from the *sixteenth* time slot in Figure 10-5. Similarly, in the *second* program, the following statements are used to find the answer $\{v_2\}$: "OPENQASM 2.0; include "qelib1.inc"; qreg q[5]; creg c[5]; x q[2]; x q[0]; h q[3]; h q[4]; h q[0]; h q[2]; cx q[4],q[2]; tdg q[2]; cx q[3],q[2]; t q[2]; cx q[4],q[2]; tdg q[2]; cx q[3],q[2]; t q[4]; t q[2]; cx q[3],q[4]; h q[2]; t q[3]; tdg q[4]; cx q[3],q[4]; cx q[4],q[2]; x q[2]; cx q[2],q[0]; x q[2]; cx q[4],q[2]; cx q[3],q[4]; tdg q[4]; t q[3]; h q[2]; cx q[3],q[4]; t q[2]; t q[4]; cx q[3],q[2]; tdg q[2]; cx q[4],q[2]; t q[2]; cx q[3],q[2]; tdg q[2]; cx q[4],q[2]; h q[2]; h q[4]; x q[3]; x q[4]; h q[4]; cx q[3],q[4]; h q[4]; x q[4]; x q[3]; h q[4]; h q[3]; measure q[3] -> c[3]; measure q[4] -> c[4];".

Figures 10-7 and 10-8, respectively, show the corresponding circuits of the two programs. In Figure 10-7 on the backend *ibmqx4*, we use quantum bits q[3], q[4], q[2], q[1] and q[0] to respectively implement quantum bits $|y_1^0\rangle$, $|y_2^0\rangle$, $|o_1^1\rangle$, $|w_{1,1}{}^0\rangle$ and $|1\rangle$ in Figure 10-5. Similarly, in Figure 10-8 on the backend *ibmqx4*, we also use quantum bits q[3], q[4], q[2], q[1] and q[0] to respectively implement quantum bits $|y_1^0\rangle$, $|y_2^0\rangle$, $|o_1^1\rangle$, $|w_{2,1}{}^0\rangle$ and $|1\rangle$ in Figure 10-6.

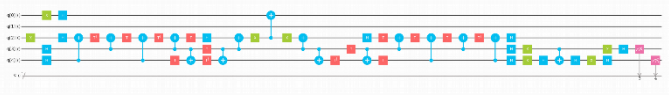

Fig. 10-7: The corresponding circuits of the first program to find the answer $\{v_1\}$.

Fig. 10-8: The corresponding circuits of the second program to find the answer $\{v_2\}$.

We use the command "simulate" to execute the two circuits in Figure 10-7 and Figure 10-8 on the target device, which is the backend *Simulator*. Figure 10-9 and Figure 10-10, respectively, show the two measured results. In Figure 10-9, we obtain the state 01000 with the probability 1.000. Because the value of q[4] is 0 and the value of q[3] is 1, we obtain the first answer $\{v_1\}$ with the probability 1.000. Similarly, in Figure 10-10, we obtain the state 10000 with the probability 1.000. The value of q[4] is 1 and the value of q[3] is 0, so we obtain the second answer $\{v_2\}$ with the probability 1.000.

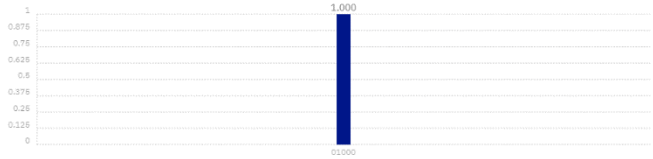

Fig. 10-9: The measured result of finding the answer $\{v_1\}$ on the backend Simulator.

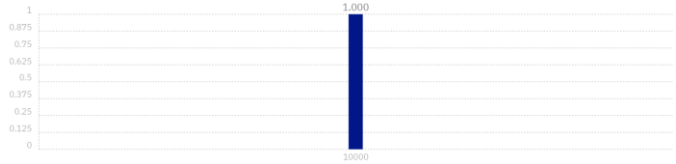

Fig. 10-10: The measured result of finding the answer $\{v_2\}$ on the backend Simulator.

We use the command "Run" to execute the two circuits in Figures 10-7 and 10-8 on *real processors* of the backend *ibmqx4*. Figure 10-11 and Figure 10-12, respectively, show the two measured results. In Figure 10-11, we obtain the state 01000 with the probability 0.541 or the state 00000 with the probability 0.154 or the state 10000 with the probability 0.112 or the state 11000 with the probability 0.217. This result is caused by the *noise* in *real processors* of the backend *ibmqx4*. Because for the state 01000 with the probability 0.541 the value of q[4] is 0 and the value of q[3] is 1, we obtain the first answer $\{v_1\}$ with the probability 0.541. Similarly, in Figure 10-12, we obtain the state 10000 with the probability 0.258 or the state 00000 with the probability 0.163 or the state 01000 with the probability 0.244 or the state 11000 with the probability 0.359. Similarly, this result is caused by the *noise* in *real processors* of the backend *ibmqx4*. Although the state 11000 has the higher probability 0.359, it encodes the set $\{v_2, v_1\}$ which is not an independent-set. Therefore, we do not select it as the answer. For the state 10000 with the probability 0.258, the value of q[4] is 1 and the value of q[3] is 0, so we obtain the second answer $\{v_2\}$ with the probability 0.258.

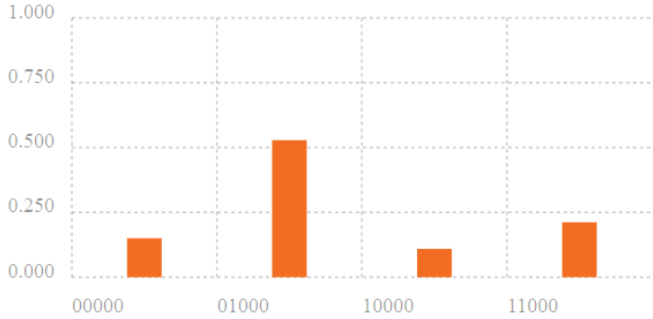

Fig. 10-11: The measured result of finding the answer {v1} on real processors of the backend ibmqx4.

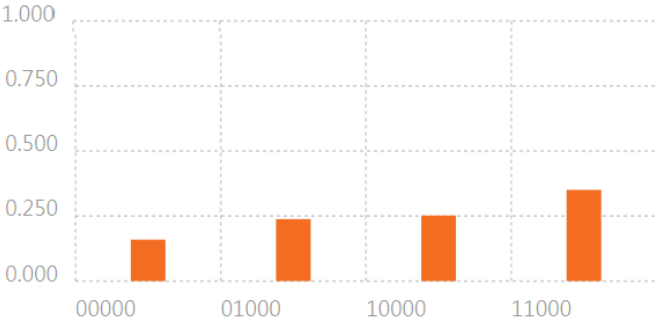

Fig. 10-12: The measured result of finding the answer {v2} on real processors of the backend ibmqx4.

XI. EXPERIMENTAL RESULTS OF FINDING THE MAXIMUM-SIZED INDEPENDENT SETS IN A GRAPH WITH THREE VERTICES AND TWO EDGES

In Figure 10-13, graph G^2 consists of three vertices and two edges. The independent sets in G^2 are $\{v_2, v_3\}$, $\{v_1\}$, $\{v_2\}$, $\{v_3\}$ and $\{\}$, which is an empty set. The *maximum-sized* independent set for G^2 is $\{v_2, v_3\}$. We write the third program in open quantum assembly language version 2.0 to find the *maximum-sized* independent set $\{v_2, v_3\}$ of graph G^2. Figure 10-14 is the corresponding quantum circuit.

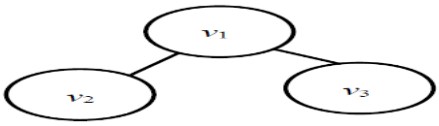

Fig. 10-13: Graph G^2 with three vertices and two edges.

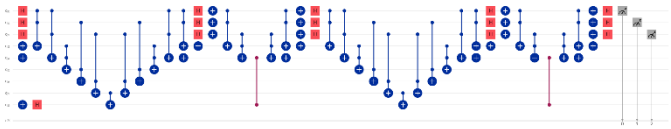

Fig. 10-14: The corresponding quantum circuit of the third program to find the answer $\{v_2, v_3\}$.

The third program labels the amplitude of the answer(s) by (-1) and amplifies the amplitude of the answer(s) twice. It specifies the four statements "OPENQASM 2.0; include "qelib1.inc"; qreg q[9]; creg c[3];" to declare nine quantum bits with the initial state |0> and three classical bits with the initial value 0. Quantum bit q[2] encodes vertex v_3, quantum bit q[1] encodes vertex v_2 and quantum bit q[0] encodes vertex v_1. Next, we use the seven statements "h q[0]; h q[1]; h q[2]; x q[8]; h q[8]; x q[3]; x q[4];" to generate all possible solutions and to set the initial state of these auxiliary quantum bits.

Then, we use the eleven statements "ccx q[0], q[1], q[3]; ccx q[0],q[2],q[4]; ccx q[3],q[4],q[5]; ccx q[1],q[5],q[6]; ccx q[2],q[6],q[7]; cx q[7],q[8]; ccx q[2],q[6],q[7]; ccx q[1],q[5],q[6]; ccx q[3],q[4],q[5]; ccx q[0],q[2],q[4]; ccx q[0],q[1],q[3];" to label the amplitude of the answer(s) by (-1). Next, we use the statements "h q[0]; h q[1]; h q[2]; x q[0]; x q[1]; x q[2]; x q[3]; x q[4]; ccx q[0],q[1],q[3]; ccx q[3],q[2],q[4]; cz q[4],q[8]; ccx q[3],q[2],q[4]; ccx q[0],q[1],q[3]; x q[0]; x q[1]; x q[2]; x q[3]; x q[4]; h q[0]; h q[1]; h q[2];" to execute the amplification of the amplitude of the answer(s).

Next, we use the eleven statements "ccx q[0], q[1], q[3]; ccx q[0],q[2],q[4]; ccx q[3],q[4],q[5]; ccx q[1],q[5],q[6]; ccx q[2],q[6],q[7]; cx q[7],q[8]; ccx q[2],q[6],q[7]; ccx q[1],q[5],q[6]; ccx q[3],q[4],q[5]; ccx q[0],q[2],q[4]; ccx q[0],q[1],q[3];" to label the amplitude of the answer(s) by (-1). Then, we use the statements "h q[0]; h q[1]; h q[2]; x q[0]; x q[1]; x q[2]; x q[3]; x q[4]; ccx q[0],q[1],q[3]; ccx q[3],q[2],q[4]; cz q[4],q[8]; ccx q[3],q[2],q[4]; ccx q[0],q[1],q[3]; x q[0]; x q[1]; x q[2]; x q[3]; x q[4]; h q[0]; h q[1]; h q[2];" to execute the amplification of the amplitude of the answer(s). And finally, we use the three statements "measure q[0] -> c[0]; measure q[1] -> c[1]; measure q[2] -> c[2];" to complete the measurement of the answer(s).

We use the command "simulate" to execute the quantum circuit in Figure 10-14 on the target device, which is the backend *Simulator*. Figure 10-15 shows the measured results for the third program. In Figure 10-15, we obtain the state 110 with the highest probability 0.55. Because the value of q[2] is 1, the value of q[1] is 1 and the value of q[0] is 0, we obtain the answer $\{v_2, v_3\}$ with the probability 0.55.

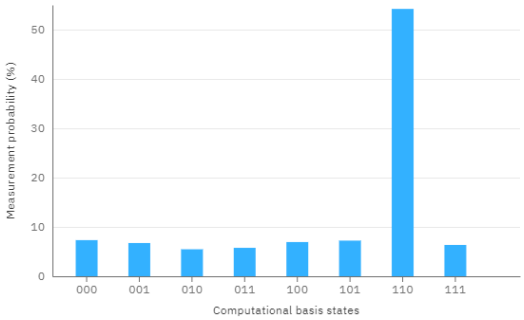

Fig. 10-15: The measured result of finding the answer $\{v_2, v_3\}$ on the backend Simulator.

XII. Conclusion

Many information processing and computing problems can be traced back to finding an extremum of a database or a cost function. Durr–Hoyer's algorithm and Ahuja–Kapoor's algorithm, which are rather useful extensions of the quantum search algorithm, are designed to find the minimum/maximum point of an unsorted database or a cost function. It is indicated in [31] that many famous quantum algorithms for finding the minimum/maximum point of an unsorted database or a cost function provide the extreme value efficiently in terms of the expected value; thus, no reasonable upper bound for the number of required elementary steps can be given. For improving the performance of Durr–Hoyer's and Ahuja–Kapoor's algorithm, quantum existence testing which integrates quantum counting and binary search [31] is proposed. The independent set problem for any graph G with m edges and n vertices is to find the maximum-sized independent set with the maximum number of vertices in an unsorted database or a cost function with 2^n subsets of vertices. However, in **Lemma 5-6**, we show that Durr–Hoyer's algorithm, Ahuja–Kapoor's algorithm and quantum existing testing cannot solve the independent set problem for any graph G with m edges and n vertices.

Lemma 4-1 to **Lemma 4-2** show that the independent set problem for any graph G with n vertices and m edges can be solved by the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m) with $O(n^2 + m)$ biological operations, $O(2^n)$ DNA strands, $O(n)$ tubes and the longest DNA strand, $O(n)$. **Lemma 5-1** to **Lemma 5-5** show that the same problem can be solved with a quadratic speed-up by **Algorithm 5-2** and **Algorithm 5-1** which implement the straightforward Boolean circuits generated from the molecular algorithm **Solve-independent-set-problem**(Y_0, n, m). In **Lemma 6-1** to **Lemma 6-4**, we show that **Algorithm 5-2** and **Algorithm 5-1** give a quadratic speed-up which is the *best* speed-up *known* for dealing with the independent set problem for any graph G with n vertices and m edges. For solving the same problem, the time complexity of the worst-case for the best classical algorithm known [52] is still $O(2^n)$. To the best of our knowledge, this is an alternative approach to the currently available best method for solving the same problem.

Furthermore, in **Lemma 7-1**, we demonstrate that mathematical solutions of molecular solutions for solving the same problem are a unit vector in the finite-dimensional Hilbert space. In **Lemma 8-1**, we show that the process of a reduction among NP-complete problems not only cannot speed up the performance of quantum algorithms but, to the contrary, slows it down. Therefore, reduction among NP-complete problems is *useless* and each NP-complete problem has its own best quantum algorithm. This means that using standard reductions followed by the proposed quantum-molecular algorithm does not necessarily lead to the best quantum algorithm for a problem in NP. Furthermore, in **Lemma 8-2**, we show that the proposed quantum-molecular algorithm with a quadratic speed-up for solving the independent set problem in a graph G with n vertices and m edges is not the best or optimal quantum algorithm. In **Lemma 9-1**, we demonstrate that for solving the element distinctness problem with an input of n bits, the proposed quantum-molecular algorithm improves the quantum lower bound of $\Omega(2^{n\times\frac{2}{3}})$ queries with a quantum walk algorithm to $\Omega(\sqrt{\frac{2^n}{2}})$ queries.

Acknowledgment

This work is supported by the National Science Foundation of the Republic of China under MOST 105-2221-E-151-040- . The authors would like to thank Dr. Amos, who is the author of the fourth reference [45], for providing valuable information on Subsection B entitled "*Introduction and Implementation of Biological Molecular Operations*" in Section IV. The authors also would like to thank Dr. Qi Yu, Professor Peng, Professor Feng and Professor Li for their valuable comments on reducing the quantum circuit that computes the number of vertices in each legal independent set.

References

[1] Feynman, R.P. In Miniaturization. In Gilbert, D.H. ed., Reinhold Publishing Corporation, New York, pp. 282-296, 1961.

[2] L. Adleman, "Molecular Computation of Solutions to Combinatorial Problems". *Science*, vol. 266, pp. 1021-1024, 1994.

[3] Feynman, R. Simulating Physics with Computers. *International Journal of Theoretical Physics*, *21*(6/7), pp. 467-488, 1982.

[4] Turing, A.M. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *s2-42*(1), pp. 230-265, 1937.

[5] Benioff, P. Quantum Mechanical Models of Turing Machines That Dissipate No Energy. *Physical Review Letter*, *48*, pp. 1581-1585, 1982.

[6] Deutsch, D. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, *400*, pp. 97-117, 1985.

[7] Garey, M.R. and Johnson, D.S. *Computer and intractability*: *A Guide to the Theory of NP-Completeness*. W. H. Freeman Company, New York, 1979.

[8] Bennett, C.H., Bernstein, E., Brassard, G. and Vazirani, U.V. Strengths and Weakness of Quantum Computing. *SIAM Journal on Computing*, *26*(5), pp. 1510-1523, 1997.

[9] Karp, R. On the Computational Complexity of Combinatorial Problems. *Networks*, *5*, pp. 45-68, 1975.

[10] Boneh, D., Dunworth, C. and Lipton, R.J. Breaking DES Using a Molecular Computer. In *Proceedings of the 1st DIMACS Workshop on DNA Based Computers*, (Princeton University), American Mathematical Society. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, pp. 37-66, 1996.

[11] Adleman, L.M., Rothemund, P.W., Roweis, S. and Winfree, E. On Applying Molecular Computation to the Data Encryption Standard. In *The 2nd annual workshop on DNA Computing*, (Princeton University), American Mathematical Society. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 31-44, 1999.

[12] Chang, W.-L., Ho, M. and Guo, M. Fast Parallel Molecular Algorithms for DNA-based Computation: Factoring Integers. *IEEE Transactions on Nanobioscience*, 4(2), pp. 149-163, 2005.

[13] Chang, W.-L. Fast Parallel DNA-based Algorithms for Molecular Computation: Quadratic Congruence and Factoring Integers. *IEEE Transactions on Nanobioscience*, Volume 11, No. 1, pp. 62-69, 2012.

[14] Lipton, R. DNA Solution of Hard Computational Problems. *Science*, 268, pp. 542-545, 1995.

[15] Adleman, L. M. On Constructing a Molecular Computer. In Lipton, R. and Baum, E. eds. *DNA Based Computers*, American Mathematical Society. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 1-21, 1996.

[16] Yeh, C.-W., Chu, C.-P. and Wu, K.-R. Molecular Solutions to the Binary Integer Programming Problem based on DNA computation. *Biosystems*, *83*(1), pp. 56-66, 2006.

[17] Ho, M. Fast Parallel Molecular Solutions for DNA-based Supercomputing: the Subset-product Problem. *BioSystems*, 80(3), pp. 233–250, 2005.

[18] Henkel, C.V., Bäck, T., Kok, J.N., Rozenberg, G. and Spaink, H.P. DNA Computing of Solutions to Knapsack Problems. *Biosystems*, 88(1-2), pp. 156-162, 2007.

[19] Chang, W.-L. Fast Parallel DNA-based Algorithms for Molecular Computation: the Set-Partition Problem. *IEEE Transactions on Nanobioscience*, Volume 6, No. 1, pp. 346-353, 2007.

[20] Yeh, C. and Chu, C. Molecular verification of rule-based systems based on DNA computation. *IEEE Transactions on Knowledge and Data Engineering*, *20*(7), pp. 965-975, 2008.

[21] Chang, W.-L. and Vasilakos, A.V. DNA Algorithms of Implementing Biomolecular Databases on a Biological Computer. *IEEE Transactions on Nanobioscience*, Volume 14, No. 1, pp. 104-111, 2015.

[22] Chang, W.-L., Vasilakos, A.V. and Ho, M. S.-H. The DNA-Based Algorithms of Implementing Arithmetical Operations of Complex Vectors on a Biological Computer. *IEEE Transactions on Nanobioscience*, Volume 14, No. 8, pp. 1-8, 2015.

[23] Woods D., Doty D., Myhrvold C., Hui J., Zhou F., Yin P. and Winfree E. Diverse and Robust Molecular Algorithms Using Reprogrammable DNA self-assembly. *Nature*, *volume* 567, pages 366-372, March 21, 2019.

[24] Ren, X. M., Wang, X. M., Wang, Z. C. and Wu, T. H. Parallel DNA Algorithms of Generalized Traveling Salesman Problem-Based Bioinspired Computing Model. *International Journal of Computational intelligence Systems*, Volume 14 Issue 1, pp. 228-237, 2021.

[25] Xu, J. et al. A DNA Computing Model for the Graph Vertex Coloring Problem Based on a Probe Graph. *Engineering*, vol. 4, no. 1, 2018, pp. 61–77.

[26] Wang, Z. C. et al. A Novel Bio-Heuristic Computing Algorithm to Solve the Capacitated Vehicle Routing Problem Based on Adleman-Lipton Model. *BioSystems*, vol. 184, 2019, p. 103997.

[27] Deutsch, D. and Jozsa, R. Rapid Solutions of Problems by Quantum Computation. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 439, pp. 553-558, 1992.

[28] Shor, P.W. Algorithm for Quantum Computation: Discrete Logarithm and Factoring Algorithm. In *Proceedings of the 35th Annual IEEE Symposium on Foundation of Computer Science*, (Santa Fe, NM, USA), pp. 124-134, 1994.

[29] Grover, L. K. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, (Philadelphia, PA, USA), ACM, pp. 212-219, 1996.

[30] Nielsen, M.A. and Chuang, I.L. *Quantum Computation and Quantum Information*. Cambridge University Press, New York, NY, 2000.

[31] Imre, S. and Balazs, F. *Quantum Computation and Communications*: *An Engineering Approach*. John Wiley & Sons Ltd., UK, 2007.

[32] Lipton R. J. and Regan K. W. *Quantum Algorithms via Linear Algebra: a Primer*. The MIT Press, 2014, **ISBN** 978-0-262-02839-4, 2014.

[33] Aaronson, S. and Shi, Y. Quantum Lower Bounds for the Collision and the Element Distinctness Problems. *Journal of the ACM*, 51: pp. 595-605, 2004.

[34] Huo, W.Y. and Long, G.L. Entanglement and Squeezing in Solid-State Circuits. *New Journal of Physics*, 10, pp. 1-11, 2008.

[35] Yang, W.-L., Wei, H., Chen, C.-Y. and Feng, M. Implementation of a many-qubit Grover search with Trapped Ultracold Ions. *Journal of the Optical Society of America B*, 25(10), pp. 1720-1727, 2008.

[36] Long, G.L. and Xiao, L. Experimental Realization of a Fetching Algorithm in a 7-qubit NMR Liouville Space Computer. *Journal of Chemical Physics*, 119, pp. 8473-8481, 2003.

[37] Boneh, D. and Lipton, R.J. Quantum Cryptanalysis of Hidden Linear Functions. In *CRYPTO '95*, Lecture Notes in Computer Science, Springer-Verlag, pp. 424-437, 1995.

[38] Lukac, M. and Perkowski, M. Evolutionary Approach to Quantum Symbolic Logic Synthesis. In the 2008 IEEE Congress on Evolutionary Computation within 2008 IEEE World Congress on Computational Intelligence, (Hong Kong, China), IEEE, pp. 3374-3380, 2008.

[39] Moylett, D. J., et al. Quantum Speedup of the Traveling-Salesman Problem for Bounded-Degree Graphs. *Physical Review A*, vol. 95, no. 3, 2017, p. 32323.

[40] Chang, W.-L., et al. Quantum Speedup in Solving the Maximal-Clique Problem. *Physical Review A*, vol. 97, no. 3, 2018, p. 32344.

[41] Pelofske, E., et al. Solving Large Minimum Vertex Cover Problems on a Quantum Annealer. *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 76–84.

[42] Arute F., Arya K., Babbush R., Bacon D., Bardin J. C., Barends R., Biswas R., Boixo S., Fernando G., Brandao S. L., Buell D. A., Burkett B., Chen Y., Chen Z., Chiaro B., Collins R., Courtney W., Dunsworth A., Farhi E., Foxen B., Fowler A., Gidney C., Giustina M., Graff R., Guerin K., Habegger S., Harrigan M. P., Hartmann M. J., Ho A., Hoffmann M., Huang T., Humble T. S., Isakov S. V., Jeffrey E., Jiang Z., Kafri D., Kechedzhi K., Kelly J., Klimov P. V., Knysh S., Korotkov A., Kostritsa F., Landhuis D., Lindmark M., Lucero E., Lyakh D., Mandrà S., McClean J. R., McEwen M., Megrant A., Mi X, Michielsen K., Mohseni M., Mutus J., Naaman O., Neeley M., Neill C., Niu M. Y., Ostby E., Petukhov A., Platt J. C., Quintana C., Rieffe E. G., Roushan P., Rubin N. C., Sank D., Satzinger K. J., Smelyanskiy V., Sung K. J., Trevithick M. D., Vainsencher A., Villalonga B., White T., Yao Z. J., Yeh P., Zalcman A., Neven H. and Martinis J. M. Quantum Supremacy Using a Programmable Superconducting Processor. *Nature*, *volume* 574, pages 505-510, 23 October 2019.

[43] Silva V. *Practical Quantum Computing for Developers: Programming Quantum Rigs in the Cloud using Python, Quantum Assembly Language and IBM Q Experience*. Apress, December 13, 2018, ISBN-10: 1484242173 and ISBN-13: 978-1484242179, 2018.

[44] Johnston, E. R., Harrigan N. and Gimeno-Segovia M. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, Inc., ISBN-13: 978-1492039686, ISBN-10: 1492039683, 2019.

[45] Amos, M. *Theoretical and Experimental DNA Computation*. Springer, ISBN-13: 978-3540657736, ISBN-10: 3540657738, April 2006.

[46] Chang, W.-L. and Vasilakos, A.V. *Molecular Computing*: *Towards a Novel Computing Architecture for Complex Problem Solving*, Springer, ISBN-13: 978-3319051215, ISBN-10: 3319051210, June 2014,

[47] Boyer, M., Brassard, G., Hoeyer, P. and Tapp, A. Tight bounds on quantum searching. *Fortsch. Physical*, 46, pp. 493-506, 1998.

[48] Durr C. and Hoyer P. A Quantum Algorithm for Finding the Minimum. arXiv: quant-ph/9607014, 1996

[49] Ahuja A. and Kapoor S. A Quantum Algorithm for finding the Maximum. arXiv: quant-ph/9911082, 1999.

[50] Cook, S. The Complexity of Theorem Proving Procedures. Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151–158,

1971.

[51] Childs A. M. and Eisenberg J. M. Quantum algorithms for subset finding. *Journal of Quantum Information & Computation*, Volume 5 Issue 7, pp. 593-604, 2005.

[52] Xiao M. and Nagamochi, H. Exact algorithms for maximum independent set. *Information and Computation*, Volume 255, pp. 126–146, 2017.